Motion Controller für ISA-, PCI-Bus, PC/104 Embedded Systeme und Standalone Operation

Benutzer- und Programmierhandbuch für POSYS® 8xx-B/9xx/18xx/19xx

POSYS Motion Control GmbH & Co.KG
Schulstr. 8 • D-92256 Hahnbach
Telefon +49 9664 9546520 • Fax +49 9664 9546521
E-mail: sales@halbeck.com
Url: www.halbeck.com

© POSYS Motion Control GmbH & Co.KG, 2013

Verzeichnis

Benutzerhandbuch 10
- Impressum 11
- Sicherheits- & Gewährleistungshinweis 12
- Einleitung POSYS® 18xx/19xx 13
- Einleitung POSYS® 7xx/8xx-B/9xx 14
- Systemüberblick 15
- Modelle 16
- Änderungen zu vorhergehenden Handbüchern 21
- Hardware 25
  - Hardware & Driver Installation 26
    - Hardware & Treiber Installation: WinDriver 27
    - Hardware & Treiber Installation: Rapid Driver 30
  - POSYS® Versionsspezifikationen 32
    - POSYS® 8xx-B 33
      - Versionen 8xx-B 33
      - Einstellbare Komponenten 33
      - Serielle Schnittstelle 35
      - Bürstenbehaftete Servomotor Pinout Beschreibung 36
      - Schrittmotor Pinout Beschreibung 38
      - Analog Eingang Pinout Beschreibung 39
    - POSYS® 18xx 41
      - Versionen 18xx 41
      - Einstellbare Komponenten 41
      - SYNC IO 45
      - Serielle Schnittstelle 45
      - CAN 2.0B 45
      - Motor Mode Konfiguration; POSYS® 18xx 46
      - Bürstenbehaftete Servomotor Pinout Beschreibung 47
      - Bürstenlose Servo- und Mikroschrittmotor Pinout Beschreibung 49
      - Schrittmotor Pinout Beschreibung 52
      - Analog Eingang Pinout Beschreibung 54
      - Verwendung des On-Board Dual Port RAMs mit der POSYS® 1800 55
    - POSYS® 9xx 59
      - Versionen 9xx 59
      - Einstellbare Komponenten 59
      - Serielle Schnittstelle 61
      - Bürstenbehaftete Servomotor Pinout Beschreibung 62
      - Schrittmotor Pinout Beschreibung 63
      - Analog Eingang Pinout Beschreibung 65
    - POSYS® 19xx 67
      - Versionen 19xx 67
      - Einstellbare Komponenten 67
      - SYNC IO 68
      - Serielle Schnittstelle 69
      - CAN 2.0B 69
      - Motor Mode Konfiguration; POSYS® 19xx 70
      - Bürstenbehaftete Servomotor Pinout Beschreibung 71
      - Bürstenlose Servo- und Mikroschrittmotoren Pinout Beschreibung 72
      - Schrittmotor Pinout Beschreibung 75
      - Analog Eingang Pinout Beschreibung 77
  - GP CON Steckverbinder 78
    - GP CON Steckverbinder 78
  - Interconnect Module 79
    - IO700/800-1 79
    - IO700/800-2 82
    - OPTON CON 86
  - Verbindungsübersicht - Motorverstärker 89

Überblick 89
POSYS® 7xx 89
POSYS® 8xx-B/18xx 90
POSYS® 9xx/19xx 91
Elektrische Spezifikationen 95
Elektrische Spezifikationen 95
Usability 97
Control Modules for POSYS® 1800/1900 mit Firmware Version >2.0 98
Control Flow Überblick 99
Aktivieren und Deaktivieren von Control Modules 100
Deaktivieren und Aktivieren des Position Loop Modules 101
Position Loop Werte lesen 102
Profilerstellung 103
Bahnen, Profile und Parameter 103
Definition der Parameter 103
Trapezfömiger Punkt-zu-Punkt Profil 104
Smoothing Factor 106
S-Kurven Punkt-zu-Punkt Profil 110
Geschwindigkeitsmodus 112
Electronic Gear Profile 113
External Profile Mode 114
Das SetStop Kommando 116
Motor Mode 117
Zykluszeit 117
Parameter Update 120
Parameter Buffering 120
Breakpoints 122
Breakpoints 122
Einen Breakpoint definieren; Übersicht 122
Breakpoint Trigger 122
Breakpoint Aktionen 123
Breakpoint Beispiele 124
Der Servo Loop 126
Überblick 126
PID Regelalgorithmus 126
Dual Encoder Support 128
BiQuad Ausgangsfilter 129
Output Limit 132
Motor Bias 133
Closed Loop and Open Loop Control Modes 133
Piezo (Keramik) Motorfunktionen 135
Status Register 138
Überblick 138
Event Status Register 138
Instruktionsfehler 139
Activity Status Register 139
Signal Status Register 140
Signal Sense Mask 141
Überwachung des Bewegungsverhaltens 142
Host Interrupts 142
In-Motion Indikator 143
Schleppabstandsfehler 143
Motion Complete Indikator 144
Settled Indikator 145
Trace Capture 146
Tracking Window 152
SetEventAction Prozess 153
Hardware Signale 155
Hardware Signale; Überblick 155
Endschalter 155
AxisOut Output 156

AxisIn Input 157
Analog Eingang 157
Der SYNC IO - Multiple Controller Synchronization 157
Motoranbindung 159
Überblick 159
Inkrementaler Enkodereingang 159
Aktuelle Positionsregister 159
Digitale Filterung 160
High Speed Position Capture 160
Parallelwort Positionsrückmeldung 160
Multi-Turn Systeme 161
Parallel-word Device Interfacing 162
Parallel Device I/O 163
Peripheral Device Read 165
Peripheral Device Write 167
Motorausgabesignale 168
Sign/Magnitude Impulsbreiten Modulierte Signalausgabe 169
50/50 Impulsbreiten Modulierte Signalausgabe 170
16 Bit DAC Signalausgabe 170
Hostkommunikation 172
Überblick 172
Paralleler Kommunikationsport 172
Serieller Kommunikationsport 172
Controller Area Network CAN2.0B 176
Konfiguration der CAN2.0B Schnittstelle 177
CAN Ereignis Benachrichtigung 177
Elektrisches Diagramm für CAN Schnittstelle 178
Verwendung des On-Board Speichers 179
Speicherkonfiguration 179
Memory Page Pointer 180
On-Board Speicher Signal Dekodierung 180
On-Board Memory Read 180
On-Board Memory Write 180
On-Board Memory Buffer 181
On-Board Speicherkommandos 181
Sinus-Kommutierung (POSYS® Achsen in bürstenlosem Modus) 183
Überblick 183
Auswahl vom 1-Phasen Signalausgabemodus mit der POSYS® 183
Signalformen für die Kommutierung 184
Kommutierungsparameter 185
Indextakt bestimmen 185
Kommutierungsfehlererkennung 186
Phaseninitializierung 186
Programmierung der Phaseninitializierung 189
Justierung der Phasenwinkel 190
Encoder Pre-Scalar 191
Konfiguration der Motorsignalausgabe 192
Open Loop Stepper Control 194
Überblick 194
Trajectory Control Units 194
Enkoderrückmeldung 194
Stall Detection 195
Takt & Richtungssignalgenerierung 196
Pulse Generation Control 197
AtRest Indikator 197
Mikroschritt Sinus/Cosinus Motorausgangssignal 198
DAC und PWM Motorausgabe (POSYS® Mikroschrittmodus) 198
Motorsignalausgabe Interpretation 198
PMW Dekodierung 198
Mikroschritt Sinus/Cosinus Motorausgangssignale 199
Mikroschrittausgangssignale 200

Motorsignalausgangskontrolle 200
AC Induction Motor Control 201
Command Summary 201
Motor Treiber Konfigurationen 201
Programmierhandbuch 204
Wie dieses Handbuch verwendet wird 204
Kommandos 205
Liste der verfügbaren Kommandos 206
Karteninitializierung und -adressierung 210
InitializePOSYS 211
OpenDevice/CloseDevice 212
DetectPosysDevices 214
RegisteredPosysDevices 215
Breakpoints und Interrupts 216
ClearInterrupt 216
GetInterruptAxis 217
Set/GetBreakpoint 218
Set/GetBreakpointUpdateMask 220
Set/GetBreakpointValue 221
Set/GetInterruptMask 223
Kommutierung 224
GetPhaseCommand 224
InitializePhase 225
Set/GetCommutationMode 226
Set/GetNumberPhases 227
Set/GetPhaseAngle 228
Set/GetPhaseCorrectionMode 229
Set/GetPhaseCounts 230
Set/GetPhaseInitializeMode 231
Set/GetPhaseInitializeTime 232
Set/GetPhaseOffset 233
Set/GetPhasePrescale 234
Digital Servo Filter 235
ClearPositionError 235
DACOutputs 236
GetAmpEnable 237
GetDerivative 238
GetIntegral 239
GetPositionError 240
GetPositionLoopValue 241
SetAmpDisable 242
SetAmpEnable 243
Set/GetAutoStopMode 244
Set/GetAuxiliaryEncoderSource 245
Set/GetBiQuadCoefficient 246
Set/GetDerivativeTime 248
Set/GetIntegrationLimit 249
Set/GetKaff 250
Set/GetKd 251
Set/GetKi 252
Set/GetKout 253
Set/GetKp 254
Set/GetKvff 255
Set/GetMotorLimit 256
Set/GetPositionErrorLimit 257
Set/GetPositionLoop 258
Set/GetProfileKp 261
Spezielle Piezo-Motorfunktionen 262
Set/GetpzIntLim 263
Set/GetpzIntLimStl 264
Set/GetpzKaff 265

Set/GetpzKd 266
Set/GetpzKdPeriod 267
Set/GetpzKi 268
Set/GetpzKp 269
Set/GetpzKvff 270
Set/GetpzKvff2 271
Set/GetpzOffsetNeg 272
Set/GetpzOffsetPos 272
Set/GetpzWindow 274
Set/GetpzWindow2 275
Enkoder 276
AdjustActualPosition 276
GetAbsoluteSSIPosition 277
GetActualVelocity 278
GetCaptureValue 279
GetSSIVersion 280
Read/WriteSSIFrequency 281
Read/WriteSSIRegister 282
Read/WriteSSIResolution 283
ResetSSI 284
SetActualToAbsolutePosition 285
Set/GetActualPosition 286
Set/GetActualPositionUnits 287
Set/GetCaptureSource 288
Set/GetEncoderModulus 289
Set/GetEncoderSource 290
Set/GetEncoderToStepRatio 291
On-Board RAM 292
DisableDPRam 293
EnableDPRam 294
ReadBuffer 294
ReadDPRamLong 296
Set/GetBufferFunction 297
Set/GetBufferLength 298
Set/GetBufferReadIndex 299
Set/GetBufferStart 300
Set/GetBufferWriteIndex 301
WriteBuffer 302
WriteDPRAMLong 303
Motor Output 304
GetActiveMotorCommand 305
GetCurrentMotorCommand 305
Set/GetHoldingCurrent 307
Set/GetMotorBias 308
Set/GetMotorCommand 309
Set/GetMotorMode 310
Set/GetMotorType 311
Set/GetOutputMode 312
Set/GetPWMFrequency 313
Set/GetStepRange 314
Operaing Mode, Event und Update Control 315
GetActiveOperatingMode 316
MultiUpdate 317
RestoreOperatingMode 318
Set/GetEventAction 319
Set/GetOperatingMode 321
Set/GetUpdateMask 323
Update 324
Profilerstellung 325
GetCommandedAcceleration 325
GetCommandedPosition 326

GetCommandedVelocity 327
Set/GetAcceleration 328
Set/GetDeceleration 329
Set/GetGearMaster 330
Set/GetGearRatio 331
Set/GetJerk 332
Set/GetPosition 333
Set/GetProfileMode 334
Set/GetStartVelocity 335
Set/GetStopMode 336
Set/GetVelocity 337
Servo Loop Kontrolle 338
GetTime 338
Set/GetAxisMode 339
Set/GetLimitSwitchMode 340
Set/GetMotionCompleteMode 341
Set/GetSampleTime 342
Set/GetSettleTime 344
Set/GetSettleWindow 345
Set/GetTrackingWindow 346
Status Register und AxisOut Indikator 347
GetActivityStatus 347
GetAxis_In_Out_Home 349
GetDriveStatus 350
GetEventStatus 351
GetSignalStatus 352
ResetEventStatus 353
Set/GetAxisOutMask 354
Set/GetAxisOutSource 356
Set/GetSignalSense 358
Traces 359
GetTraceCount 359
GetTraceStatus 360
Set/GetTraceMode 361
Set/GetTracePeriod 362
Set/GetTraceStart 363
Set/GetTraceStop 365
Set/GetTraceVariable 367
Verschiedenes 369
GetChecksum 369
GetHostIOError / GetInstructionError 370
GetProcessorStatus 371
GetVersion 372
HardReset/MCReset 373
NoOperation 376
PortInOutW 377
ReadAnalog 378
ReadIO 379
ReadPLD 380
Set/GetCANMode 381
Set/GetDiagnosticPortMode 382
Set/GetSerialPortMode 383
Set/GetSynchronizationMode 385
SetWatchDog 386
WriteIO 387
Zusätzliche MotionScript® Funktionen 388
BitReset 388
BitSet 389
BitTest 390
CompareActivity 391
CompareEvent 392

CompareSignal 393
Kommandoübersicht - ReadIO/WriteIO 394
ReadIO / WriteIO 394
Application Notes 396
Extension Port Connector J6 (POSYS® 700, ISA-Bus) 396
Stall Detection 397
Coordinated Motion with POSYS® Contouring Library 398
Coordinated Moves using POSYS® 900 Motion Controller 402
Digital IO Specification POSYS® 7xx/8xx/8xx-B/9xx 403
Digital Inputs & Outputs 404
Breakpoints 406
Error Code 410
On-Board RAM 412
Limit Switches 415
Sinusoidal Commutation and Microstep Mode Setup 418
Mixed Motor Mode 420
Multiple Cards 421
PCI ScanForPOSYS 423
Programmer´s Info 428
Resistor Networks for TTL or inverted Encoder Signals 437
Serial Interface 438
Status Read Operation 440
Synchronization of Multple Cards (POSYS® 8xx-B/9xx w/ Sync-Feature and 18xx/19xx) 442
Velocity, Acceleration and Jerk Calculation 444
MotionScript® 446
Overview; MotionScript® 446
POSYSDriverXSetup 447
Program Use 449
Examples 450
Variables 452
Reserved Words 453
MotionScript® Utility Functions 454
Other MotionScript® Functions 455
Loop Instructions 456
Writing MotionScript® Functions 457
POSYS® Functions and Procedures 458
POSYS® GUI für Windows 98/ME/2000/XP 465
Einleitung 465
Das POSYS® GUI Hauptfenster 466
Initialisierung 467
Beschreibung der weiteren Programmfunktionen 468
Controller Status Box 469
Power-On Defaults Box 469
Motor Output Mode Box 469
Filter Parameter Box 470
Verstärkerfreigabe und DACOutputs ON/OFF 471
Output Torque Box 472
Update Periods Box 472
Status Activity/Event/Signal 472
Signal Sense 473
Enter Data Box 474
Commanded und Actual Position Anzeigebox 476
SSI Rückmeldung 476
Analog Inputs 477
Single Port RAM / Dual Port RAM 479
Trace Mode 479
Setupmenü für Sinuskommutierung und Mikroschrittmotormodus 482

Benutzerhandbuch

Impressum

Handbuch
für
POSYS® Positioniersteuerung
der Serien 800-B/1800/900/1900

POSYS Motion Control GmbH & Co.KG
Schulstr. 8
D-92256 Hahnbach

Tel: +49 9664 9546520
Fax: +49 9664 9546521
E-mail: support@halbeck.com
Url: http://www.halbeck.com

Geschäftsführer: Thomas Halbeck
HRA: 3227 Amtsgericht Amberg

Revision: 7.2
Datum: 07.06.2013

¤

Sicherheits- & Gewährleistungshinweis

Gewährleistung

POSYS Motion Control GmbH & Co.KG garantiert die Leistung seiner Produkte entsprechend den zur Verfügung stehenden Spezifikationen zum Zeitpunkt des Verkaufs für einen Zeitraum von 2 Jahren. Ebenso garantiert POSYS Motion Control GmbH & Co.KG für einen Zeitraum von 2 Jahren ab Verkaufsdatum die einwandfreie Materialbeschaffenheit unter der Voraussetzung, dass die Produkte innerhalb der gesetzten Spezifikationen betrieben werden und ein normaler täglicher Gebrauch unterstellt wird, d.h. Produkte die länger als 8 Stunden täglich Verwendung finden, können einer kürzeren Garantieperiode unterliegen. Für Details setzen Sie sich mit POSYS Motion Control GmbH & Co.KG in Verbindung.
POSYS Motion Control GmbH & Co.KG erlaubt sich Änderungen an seinen Produkten vornehmen zu können, oder Produkte oder Dienstleistungen einzustellen ohne vorhergehende Information und bittet Kunden und Interessenten die möglichst neuesten Informationen zu verifizieren ehe Bestellungen plaziert werden um sicher zu stellen, dass alle Informationen aktuell und komplett sind. Alle verkauften Produkte unterliegen den erweiterten Verkaufsbedingungen der ZVEI von 2002.

Sicherheitshinweis und Haftungsausschluss

Bestimmte Applikationen in denen elektronische Komponenten verwendet werden, können ein potenzielles Risiko für Leben und Material darstellen. Unsere Produkte sind nicht entwickelt worden bzw. wir authorisieren nicht die Verwendung unserer Produkte in Gerätschaften die der Lebenserhaltung dienen oder anderen kritischen Applikationen. Die Verwendung der Produkte der Firma POSYS Motion Control GmbH & Co.KG in solchen Gerätschaften bzw. in solch kritischen Applikationen geschieht ausser Garantie und auf volles Risiko des Kunden.
Um Risiken in Verbindung mit Applikationen des Kunden zu minimieren, müssen das Design und die operativen Sicherheitsmaßnahmen durch den Kunden sichergestellt sein um Gefahren abzuwehren.
POSYS Motion Control GmbH & Co.KG übernimmt keine Haftung für Anwendungshilfe oder Kundenproduktdesign. POSYS Motion Control GmbH & Co.KG übernimmt auch keine Haftung für Produktionsausfall, Schäden oder sonstige Kosten, ausser für Kosten die für die notwendige Reparatur oder Ersatz für Produkte der Firma POSYS Motion Control GmbH & Co.KG anfallen. Weitere Ersatzansprüche können nicht geltend gemacht werden.

¤

Einleitung POSYS® 18xx/19xx

Die POSYS®-Serie ist ein hochentwickeltes Positioniersystem für bürstenbehaftete und bürstenlose Servomotoren, Hydrauliksysteme, Mikro- oder Schrittmotoren. Die Karten halber Baugröße für den PCI-Bus (POSYS® 19xx) werden in freie Steckplätze eines IBM-kompatiblen Computers gesteckt. Jede Karte ist in Konfigurationen für 1, 2, 3 oder 4 Achsen für Servo (bürstenbehaftete oder bürstenlose) und Schrittmotoren (Standard oder Mikroschritt) erhältlich (jede Achse frei-konfigurierbar oder in speziellen, abgespeckten Versionen mit eingeschränkten Möglichkeiten). Die Verwendung eines DSP zusammen mit einem ASIC sorgt zusammen mit dem 16-bit DAC-, 10-bit PWM oder Takt- & Richtungsausgangssignal für ausgezeichnete Positioniersteuerungsqualitäten. Zusammen mit den 16 frei-programmierbaren E/As gibt es außerdem noch dedizierte Eingänge (richtungsabhängige Endschalter mit einem digitalen Filter und High Speed Latch Eingänge für jede Achse). Desweiteren stehen 2 achsenspezifische E/As und für jede Achse ein Ausgang für die Verstärkerfreigabe zur Verfügung. Diese E/As können auch für eigene Zwecke verwendet werden. Programmierbare Zeitnehmer könnten auch als Zähler eingesetzt werden. Ein digitaler PID- oder PI-Filter mit KP, KL, KD, Kvff und IL und der weite, programmierbare 16-Bit-Bereich sind zusätzliche Gründe für das überlegene Verhalten dieses Reglers. Ein weiteres Plus sind die hohe 5 MHz Inkremental- oder 160 MHz-Absolutenkodereingangsfrequenz und je nach Modell die 50 bis 75 µsec Servo-Loop-Updaterate pro Achse.

Die POSYS® 18xx/19xx Serie wurde entwickelt um höchste Flexibilität zu einem erschwinglichen Preis zu gewährleisten und um Antworten für komplexe Steuerungsprobleme zu bieten sowie die Peripheriekomponenten wie die digitalen und analogen Eingänge und die programmierbaren Ausgänge auf einfache Art zu handhaben. Die dedizierten Ausgänge für In-Position, Schleppabstandsfehlererkennung und Verstärkerfreigabe signalisieren den eigentlichen Status jeder Achse. Software-Kompatibilität wird durch alle Versionen garantiert (außer anders notiert; z.B. bei motorspezifischen Unterschieden).

Der POSYS®-Regler wurde für komplexe Positioniersteuerungsanwendungen wie u.a. Jogging, Punkt-zu-Punkt Positionierung, Vektorpositionierung, Electronic Gearing und für multiple Bewegungssequenzen entworfen. S-Kurven Profile eliminieren ruckartige positive und negative Beschleunigung; dies ist besonders nützlich, wenn schwere Trägheitslasten bewegt werden. PMD's Pro-Motion Software mit einer Vielzahl unterschiedlicher textalischer und graphischer Ausgabemöglichkeiten als auch der Möglichkeit Hardcopies für Vergleichszwecke in der Performance zu erstellen, helfen dem Anwender die optimalen Filterparameter für jede mögliche Ladung und Systemzustand zu ermitteln. Pro-Motion Software hilft dem Anwender durch Beobachtung und Analyse der Reaktionen des Servosystems auf dem Monitor die optimalen Filtereinstellungen zu finden und einzustellen.

¤

Einführung POSYS® 7xx/8xx-B/9xx

Die POSYS®-7xx/8xx-B/9xx-Serie ist ein hochentwickeltes Positioniersystem für bürstenbehaftete und bürstenlose Servomotoren, Hydrauliksysteme oder Mikro- oder Schrittmotoren. Die Karten halber Baugröße für den ISA- bzw. PCI-Bus (POSYS® 7xx/8xx-B und 9xx) werden in freie Steckplätze (ISA-, PC/104 oder PCI; modellabhängig) eines IBM-kompatiblen Computers gesteckt und die POSYS® 8xx-B sind 100% PC/104 compatible Module für Embedded Systeme. Jede Karte ist in Konfigurationen für 2 oder 4 Achsen für Servo (bürstenbehaftete oder bürstenlose mit entsprechendem Treiber) oder Schrittmotoren (Standard oder Mikroschritt mit entsprechendem Treiber) erhältlich. Die Verwendung eines DSP zusammen mit einem ASIC sorgt zusammen mit dem 16-bit DAC-, 10-bit PWM oder Takt- & Richtungsausganssignal für ausgezeichnete Positioniersteuerungsqualitäten. Zusammen mit den 16 frei-programmierbaren E/As gibt es außerdem noch dedizierte Eingänge (richtungsabhängige Endschalter mit einem digitalen Filter und High Speed Latch Eingänge für jede Achse). Desweiteren stehen 2 achsenspezifische E/As und für jede Achse ein Ausgang für die Verstärkerfreigabe zur Verfügung. Diese E/As können auch für eigene Zwecke verwendet werden. Programmierbare Zeitnehmer könnten auch als Zähler verwendet werden. Ein digitaler PID- oder PI-Filter mit KP, KL, KD, Kvff und IL und der weite, programmierbare 16-Bit-Bereich sind zusätzliche Gründe für das überlegene Verhalten dieses Reglers. Ein weiteres Plus sind die hohe 5 MHz Inkremental- oder 160 MHz-Absolutenkodereingangsfrequenz und die 100 bzw. 150 µsec Servo-Loop-Updaterate pro Achse.

Die POSYS® Serie wurde entwickelt um höchste Flexibilität zu einem erschwinglichen Preis zu gewährleisten und um Antworten für komplexe Steuerungsprobleme zu bieten, die Peripheriekomponenten wie die digitalen und analogen Eingänge und die programmierbaren Ausgänge zu handhaben. Die dedizierten Ausgänge für In-Position, Schleppabstandsfehlererkennung und Verstärkerfreigabe signalisieren den eigentlichen Status jeder Achse. Software-Kompatibilität wird durch alle Versionen garantiert (außer anders notiert; z.B. bei motorspezifischen Unterschieden).

Der POSYS®-Regler wurde für komplexe Positioniersteuerungsanwendungen wie u.a. Jogging, Punkt-zu-Punkt Positionierung, Vektorpositionierung, Electronic Gearing und multiple Bewegungssequenzen entworfen. S-Kurven Profile eliminieren ruckartige positive und negative Beschleunigung; dies ist besonders nützlich, wenn man schwere Trägheitslasten bewegt. PMD's Pro-Motion mit einer Vielzahl unterschiedlicher textalischer und graphischer Ausgabemöglichkeiten als auch der Möglichkeit Hardcopies für Vergleichszwecke in der Performance zu erstellen, helfen dem Anwender die optimalen Filterparameter für jede mögliche Ladung und Systemzustand zu ermitteln. Pro-Motion Software hilft dem Anwender durch Beobachtung und Analyse der Reaktionen des Servosystems auf dem Monitor die optimalen Filtereinstellungen zu finden und einzustellen.

¤

Systemüberblick

Die Position jeder Achse wird ermittelt durch Rückmeldung unter zur Hilfenahme verschiedenartiger Rückmeldeeinheiten, entweder durch inkrementale oder Absolutwertgeber, analogen oder digitalen Konvertern, Resolver oder Laserinterferometer. Falls inkrementale Signale benutzt werden, werden die eingehenden A und B-Signale digital gefiltert, und werden dann zu einem high-speed Zähler weiter vermittelt. Falls die Parallelwortschnittstelle benutzt wird, wird eine direkte binär-verschlüsselte Position von bis zu 16 Bit von der POSYS® gelesen. Ungeachtet der Enkodereingangsmethode, werden diese Positionsinformationen benutzt um ein 32-Bit Positionswert zu erhalten.

Der Controller kalkuliert eine neue Zielposition zu jedem Zeitintervall, basierend auf dem vorgegebenen Pofilmodus und -parametern die einerseits vom Host vorgegeben wurden, aber die auch auf den gegenwärtigen Istzustand des Systems beruhen. Die Zykluszeit ist die Rate, zu der größere Anlagenparameter aktualisiert werden, wie Drehmomentsollwert, Servokompensation (falls Servoversionen benutzt werden), und einige andere Funktionen. Für die Servokontrolle wird der Ausgang des Controllers mit der eigentlichen Enkoderposition verglichen um einen 32-Bit Schleppabstandsfehler zu kalkulieren, der durch ein PID-Filter geleitet wird. Der resultierende Drehmomentsollwert wird dann von der POSYS® ausgegeben um einen externen Verstärker entweder mit dem PWM- oder DAC-Signal anzusteuern. Mit der POSYS® im Modus für bürstenbehaftete Servos ist eine On-Board Kommutierung nicht möglich und die Ansteuerung der Phasen geschieht direkt. Für bürstenlose Servomotoren ist ein Verstärker notwendig, der die Aufgabe das Kommutierungssignal zu generieren, übernehmen kann, oder die Verwendung der POSYS® im bürstenlosen Servomodus, da diese Version die Möglichkeit der On-Board Generierung der Kommutierungssignale bietet. Das Ausgangssignal des Controllers der POSYS® im Schrittmotormodus wird konvertiert um für den Schrittmotor ein entsprechendes Takt & Richtungssignal zu liefern.
Kommunikation von/zu der POSYS® Positioniersteuerung wird bewerkstelligt durch Verwendung der Mikroprozessor-ähnlichen Parallel-Bus-Schnittstelle. Dies erlaubt die Kommunikation von 16-Bit breiten Übertragungen.
Kommunikation von/zu der POSYS® wird durch Senden und Empfangen einer Sequenz aus Bytes und Words, bestehend aus kurzen Kommandos, erledigt. Diese Datenpakete enthalten ein Anweisungscodewort, das der POSYS® mitteilt, welche Aufgabe ausgeführt werden soll.
Diese Befehle werden von einem Hostmikroprozessor oder Hauptrechner gesandt, das ein Supervisorprogramm ausführt. Die POSYS® ist so konzipiert um Geschwindigkeit und andere spezielle bewegungsspezifische Aufgaben selbständig zu bewältigen und diese an die entsprechenden Stellen weiter zu leiten, während die Hostsoftware die allübergreifende Kontrolle behält.

Andere Hauptaufgabenstellungen der POSYS® umfassen:

Breakpoints
Breakpoints erlauben die Überwachung verschiedener Signale, Parameter oder Bedingungen um mit einer Bedingung verglichen zu werden. Breakpoints können programmiert werden, automatisch die POSYS®-Verhaltensweise zu regulieren, wenn dem Zustand entsprochen wird. Dies ist nützlich für Aufgabenstellungen wie "die Geschwindigkeit verändern wenn ein Signal aktiv wird".

Diagnostische Parametererfassung
Die Diagnostische Parametererfassung erlaubt Werte von bis zu vier Parametern gleichzeitig automatisch im On-Board RAM der POSYS® zur späteren Auswertung vom Host, abzuspeichern. Diese Fähigkeit macht es leicht, sehr genaue Diagramme vom Servoverhalten und Bahninformationen, etc. zu erhalten ohne den Hostrechner mit Aufgaben zur Erfassung von Real-Time Daten zu belasten.

Schleppabstandsfehler, Tracking Window, und Settle Window
Diese Funktionen erlauben die automatische Erfassung von Schleppabstandsfehlern (Abstand zw. Ist- und Sollposition). Sie sind dann von Wert wenn bestimmte Konditionen eintreten die eine besondere Handhabung bedürfen wie z.B.: "Halte die Achse an, falls ein bestimmter Wert überschritten wird." oder "definierte Bewegungssequenz ist beendet, falls die Achse sich innerhalb einer programmierten Schleppabstandsfehlerweite befindet für eine bestimmte Zeitspanne."

Endschalter
Erlaubt automatisch die Achse zu stoppen falls sich die Achse ausserhalb gesetzter Werte befindet

Andere POSYS®-Merkmale umfassen analoge Signaleingänge, Software-invertierbare digitale Signale, und Anwender-definierte I/O-Decodierung nur um ein paar zu nennen.

In den folgenden Abschnitten werden jeder dieser POSYS®-Funktionen besprochen und erklärt. Die meisten POSYS®-Funktionen sind gleich durch alle Versionen. Allerdings beschreiben einige Abschnitte dieses Handbuches Merkmale, die sich nur auf besondere Motormerkmale beziehen (Servo oder Schrittmotoren). Zum Beispiel bezieht sich die PID-Filterung nur auf die Servo-Versionen der POSYS®. Wenn dies der Fall ist, werden diese Abschnitte dann deutlich als solche gekennzeichnet.

¤

Modelle

Modelle bestückt mit Magellan Chipsatz

Bus	PCI		
# Achsen/Motor	Multiple Motor Positioniersteuerkarte individuell auf jede Achse einstellbar für Servomotoren (inkl. interne Generierung der Kommutierungssignale für bürstenlose Servos), Mikroschritt- und Schrittmotoren	Multiple Motor Positioniersteuerkarte individuell auf jede Achse einstellbar für Servomotoren (keine interne Generierung der Kommutierungssignale für bürstenlose Servos) und Schrittmotoren (Version eingestellt)	Schrittmotor Karte
2	1922	1902 (n/a)	1952
4	1924	1904 (n/a)	1954

Bus	PC/104		
# Achsen/Motor	Multiple Motor Positioniersteuerkarte individuell auf jede Achse einstellbar für Servomotoren (inkl. interne Generierung der Kommutierungssignale für bürstenlose Servos), Mikroschritt- und Schrittmotoren	Multiple Motor Positioniersteuerkarte individuell auf jede Achse einstellbar für Servomotoren (keine interne Generierung der Kommutierungssignale für bürstenlose Servos) und Schrittmotoren (Version eingestellt)	Schrittmotor Karte
2	1822	1802 (n/a)	1852
4	1824	1804 (n/a)	1854

Modelle bestückt mit Navigator Chipsatz*

Bus	ISA		PC/104			
# Achsen/Motor	Brushed Servo (nur auf Anfrage verfügbar)	Stepper (nur auf Anfrage verfügbar)	Brushed Servo	Brushless Servo with brushless servo drive	Stepper	Combination of Brushed and Brushless Servo and microstepping w/wo SYNC IO (Versionen eingestellt)
2	702	752	802-B	802-B	852-B	872-B/882-B (n/a)
4	704	754	804-B	804-B	854-B	874-B/884-B (n/a)

Bus	PCI			
# Achsen/Motor	Brushed Servo	Brushless Servo w/ brushless servo drive	Stepper	Combination of Brushed and Brushless Servo and microstepping w/wo SYNC IO (Versionen eingestellt)
2	902	902	952	972/982 (n/a)
4	904	904	954	974/984 (n/a)

Multiple Motor Positioniersteuerkarten

Multiple Motor Positioniersteuerkarten sind ausgelegt folgende Motorausgangssignale zu unterstützen:

Motorausgangssignal	Beschreibung
DAC	±10V analoges Ausgangssignal für bürstenbehaftete und bürstenlose Servomotoren mit externer Kommutierung, oder 2- und 3-Phasen bürstenlose Servomotoren mit Kommutierung auf der Karte
PWM	Sign/Magnitude impuls-breiten-moduliertes Signal für bürstenbehaftete und bürstenlose Servomotoren mit externer Kommutierung, oder 2-Phasen bürstenlose Servomotoren mit Kommutierung auf der Karte
	50/50 impuls-breiten-moduliertes Signal für bürstenbehaftete und bürstenlose Servomotoren mit externer Kommutierung, oder 2- und 3-Phasen bürstenlose Servomotoren mit Kommutierung auf der Karte
Takt & Richtung	Takt & Richtungssignal für Schrittmotoren (nur Magellan bestückte Versionen)
Mikroschritt	2-phasige Ausgangssignale pro Achse in entweder PWM oder DAC-kompatiblem Format für Mikroschrittmotoren

Jede Achse einer Multiplen Motor Positioniersteuerkarte ist individuell einstellbar für die Verwendung mit einer der folgenden Motortypen.

Bürstenbehafteter Servomotor mit ±10-Volt Drehmomentsollwertvorgabe oder PWM-Signal (sign/magnitude oder 50/50)

Die Standard-Konfiguration ist für Verwendung mit ±10-Volt, 16-Bit Drehmomentsollwertvorgabe für Servomotoren. Die POSYS® erreichen ihre überlegene Präzision durch Verwendung eines 16-Bit Motor-DACs und einem hochentwickelten, skalierbaren PID-Filter, welches Velocity und Acceleration Feedforward und BIAS als Ausstattungsmerkmale mitbringt, als auch Integration Limit, einstellbare Derivative Sampling Time und eine programmierbare Motorausgangssignalbegrenzung (z.B. ±2,3V anstatt ±10V).

Diese Konfiguration kann auch in Verbindung mit bürstenlosen Servomotoren verwendet werden, falls Verstärker eingesetzt werden die das Kommutierungssignal extern generieren können. Dies ist dann notwendig wenn folgende Modelle eingesetzt werden: POSYS® 70x/80x-B/90x/182x/192x. Wobei die 182x und 192x auch intern die Möglichkeit bieten das Kommutierungssignal zu generieren (Sinuskommutierung).

Bürstenloser Servomotor mit ±10-Volt Drehmomensollwertvorgabe oder PWM (sign/magnitude oder 50/50)

Der Unterschied zu der standardmäßigen Servomotorkonfiguration ist, daß der Anwender einen Verstärker benötigt, der ein sinusförmiges Kommutierungssignal generiert wenn er die Positioniersteuerung im Standardausgabemodus verwendet (70x/80x-B/90x-B/182x/192x). Je nach Modell sind die POSYS-Regler auch konfigurierbar mit On-Board Sinus-Kommutierung um mit entsprechenden Verstärkern eingesetzt werden zu können. Dies ist bei folgenden Modellen der Fall: POSYS ® 182x/192x

Schrittmotor mit Takt & Richtungssignalen

Die POSYS® können Schrittmotoren steuern. In diesem Modus liefert der Regler zwei Signale um den Schrittmotor zu bewegen: Takt und Richtung. Für Schrittmotoroperationen benötigt der Regler keinen Enkoder, obwohl der Regler Enkodereingangssignale als Standard unterstützt, arbeitet es im Open Loop Modus. Die max. Ausgangsfrequenz ist 4,98 Mhz. Die POSYS® im Schrittmotormodus bietet die gleichen funktionalen Merkmale wie die POSYS® im Servomotormodus, aber ohne PID-Filtereinstellmöglichkeiten. Dies ist bei folgenden Modellen der Fall: POSYS® 75x/85x-B/95x/18xx/19xx

Mikroschritt mit 90 Grad/120 Grad versetztem Signalausgang für 2- und 3-Phasen-Schrittmotoren

Die POSYS-Karte unterstützt die Verwendung von 2- und 3-Phasen Schrittmotoren, dabei stellt sie eine programmierbare Mikroschrittrate von 1 bis 256 Mikroschritten/Vollschritt zur Verfügung und gibt dabei ein PWM oder DAC-kompatibles Motorausgangssignal um die Wicklungen eines Schrittmotors direkt anzusprechen. Dies ist bei folgenden Modellen der Fall: POSYS® 182x/192x.

Schrittmotorkarte

Motorausgangssignal	Beschreibung
Takt & Richtung	Takt & Richtungssignal für Schrittmotoren

Die Schrittmotorkarte ist eine kostengünstige Version der Multiplen Motor Positionisteuerkarte die allein darauf ausgelegt ist die gleiche Funktionalität zu einem erheblich günstigeren Preis anzubieten, unter der Voraussetzung, dass im System nur Schrittmotoren bzw. mit entsprechenden Treibern auch Mikroschrittmotoren zum Einsatz kommen sollen.

Kundenspezifische Lösungen

Alle POSYS® Positioniersteuerkarten lassen sich auch kundenspezifisch konfigurieren. Dies umfasst spezielle Speichermodule, über Erweiterungen für E/As (digital und analog), als auch in umgekehrte Richtung eine Abspeckung um Karten kostengünstiger zu gestalten falls gewisse Komponenten und Funktionalitäten nicht benötigt werden.

Spezifikationen

Die folgende Tabelle listet die Hauptmerkmale bzw. Unterschiede zwischen der Multiplen Motor Positioniersteuerkarte und der reinen Schrittmotorsteuerkarte auf die mit dem Magellan Chipsatz bestückt sind:

POSYS® 18xx/19xx

Spezifikation	POSYS® 182x/192x	POSYS® 180x/190x (Versionen eingestellt)	POSYS® 185x/195x
Unterstüzte Anzahl Achsen	2, 4	2, 4	2, 4
Unterstützte Motorversionen	Servo (inkl. interne Kommutierung für DC brushless), Mikroschritt, Takt & Richtung	Servo (inkl. externe Kommutierung für DC brushless), Takt & Richtung	Takt & Richtung
Ausgabeformat	PWM, DAC, Takt & Richtung	PWM, DAC, Takt & Richtung	Takt & Richtung
Parallele Kommunikation	+	+	+
Serielle Kommunikation	+	+	+
CAN 2.0B	+	+	+
Inkrementaler Enkodereingang (bis zu 8 Mhz)	+	+	+
Paralleler Enkodereingang (mit Add-On Karte; bis zu 160 Mhz; Absolutwertgeber, Resolver etc.)	+	+	+
Index & Home Eingang pro Achse	+	+	+
High Speed Latch Eingang pro Achse	+	+	+
Position Capture	+	+	+
Endschalter (2x pro Achse)	+	+	+
PWM Ausgabe	+	+	
Parallel DAC Ausgabe	+	+	
SPI DAC Ausgabe	+	+	
Interne Kommutierung für bürstenlose Servomotoren	+		
Direkte Unterstützung von Mikroschrittmotoren	+	+ (mit entsprechendem Treiber)	+ (mit entsprechendem Treiber)
Takt & Richtungssignal	+	+	+
Trapezförmige Bahngenerierung	+	+	+
Geschwindigkeitsmodus	+	+	+
S-Kurvenmodus	+	+	+
Electronic Gearing	+	+	+
On-the-Fly Änderungen von Filter-Parameter (nur Servomotoren), Geschwindigkeit, Beschleunigung, Position	+	+	+ Keine Filter-Parameter
PID Servokontrolle	+	+	
Dual Enkoder Loop	+	+	
Derivative Sampling Time	+	+	
Feedforward (Beschl. und Geschw.)	+	+	
Dual Bi-Quad Filter	+	+	
Real-Time Datenerfassung / Diagnostik (nutzt On-Board RAM)	+	+	+
Bahndaten-Upload (nutzt On-Board RAM)	+	+	+
Schleppabstandsfehlererkennung	+	+	+ (erfordert Enkoder)
In-Position	+	+	+

Spezifikation	POSYS® 182x/192x	POSYS® 180x/190x (Versionen eingestellt)	POSYS® 185x/195x
			(erfordert Enkoder)
Analogeingänge (8x 10-Bit)	+	+	+
Programmierbarer Bitausgang	+	+	+
Software invertierbare Signale	+	+	+
Anwender-spezifische E/A	+	+	+
On-Board Speicherunterstützung	+ Single Port RAM (512 KB optional) Dual Port RAM standard (32 oder 64 KB)	+ Single Port RAM (512 KB standard) Dual Port RAM optional (32 oder 64 KB)	+ Single Port RAM (512 KB standard) Dual Port RAM optional (32 oder 64 KB)
Multiple Kartensynchronsation	+ (ausser eine Achse wird im Schrittmotormodus betrieben)	+ (ausser eine Achse wird im Schrittmotormodus betrieben)	
Watchdog Timer	+	+	+
Interrupt-Behandlung	+	+	+
Softwareunterstützung für Windows NT/2000/XP/Vista/7, Linux ab Kernel 2.4; DOS-Unterstützung auf Anfrage; zusätzliche Softwarefunktionen für Interpolationsaufgaben (zirkular) auf Anfrage; z.T. kostenpflichtig	+	+	+

POSYS 7xx/8xx-B/9xx

Verfügbare Konfigurationen (ISA, PC/104, PCI; Servo oder Schritt)	4 Achsen: ISA: 704, 754 (nur auf Anfrage erhältlich) PC/104: 804-B, 854-B PCI: 904, 954 2 Achsen (nur auf Anfrage erhältlich; Mindestmengenabnahme): ISA: 702, 752 PC/104: 802-B, 852-B PCI: 902, 952
Operationsmodi	Closed loop (Drehmomentsollwertvorgabe durch Vorgabe des Servofilters; PID-Regelung) Open loop (Anweder-programmierbare Register)
Kommunikationsmodi	16/16 parallel (16 bit externer Parallelbus mit 16 bit interner Kommandowortgröße)
Positionsbereich	-2,147,483,648 to +2,147,483,647 counts
Geschwindigkeitsbereich	-32,768 to +32,767 counts/sample mit einer Auflösung von 1/65,536 counts/sample
Beschleunigungsbereich (positiv und negativ)	-32,768 to +32,767 counts/sample² mit einer Auflösung von 1/65,536 counts/sample²
Jerkbereich für S-Kurvenbeschleunigung	0 to ½ counts/sample³, mit einer Auflösung von 1/4,294,967,296 counts/sample³
Profilmodi	S-Kurve Punkt-zu-Punkt Rampe (Geschwindigkeit, Beschleunigung, Smoothing und Positionangaben); Änderung von Parametern on-the-fly während des Ablaufs nicht veränderbar Trapezförmige Punkt-zu-Punkt Rampe (Geschwindigkeit, negative/positive Beschleunigung und Positionsangaben); alle Parameter während des Ablaufs on-the-fly veränderbar Electronic Gear (Enkoder- oder Positionssollwertvorgabe dient als Vorgabe um eine oder mehrere Achsen zu bewegen mit ±16-Bit Verhältnis. Master und Slave Achsen und Gear Ratio Parameters) External Profile Mode: erlaubt die Abspeicherung abzufahrender Bahndaten im Speicher des Kontrollers. Dadurch geringe Belastung des Hostprozessors. Zudem kann parallel der Speicher für die Datenerfassung bestimmter Parameter verwendet werden für Diagnostik, Wartung oder um die Leistung zu optimieren durch Heranziehen von Vergleichswerten. OPTION: Trapezförmige Punkt-zu-Punkt Rampe mit Smoothing Faktor (on-the-fly Änderungen aller Parameter möglich; Geschwindigkeit, negative/positive Beschleunigung, und Positionsparameter)
Electronic Gear Einstellbereich	-32,768 to +32,767 mit einer Auflösung von 1/65,536 (negative and

	positive Richtung)
Filtermodi	Scalierbare PID (nicht Schrittmotorversion) + Velocity feedforward + Acceleration feedforward + Bias. Auch Integrationslimit, einstellbare Derivative Sampling Time und Begrenzung der Drehmomensollwertvorgabe
Filter Parameter Auflösung	16 Bit (Servo)
Schleppabstandsfehler-erkennung	Schleppabstandsfehlerfenster (bei erkanntem Schleppabstandsfehler (32-Bit) kann ein automatischer Stop (abrupt oder mit programmierbarer Rampe oder ignorieren) programmiert werden. Tracking Window (erlaubt das Setzen eines Flags falls eine Achse ein definiertes, programmierbares Positionsfenster überschreitet für eine programmerbare Zeitspanne nach Bewegungsende
Ausgangssignale	Servo: DAC (16 Bit) ±10V Ausgabe PWM (10 Bit Auflösung bei 20 Khz) Schritt: Takt & Richtungssignal Mikroschritt: DAC (16 Bit) ±10V Ausgabe PWM (10 Bit Auflösung bei 20 Khz)
Maximale Encoderrate	Inkremental (bis 4,98 Mhz.); OPTION: Parallel (bis 160 Mhz.)
Servoloopupdaterate	100 µsec nominal (exakte Zeit ist 102,4 µsec) pro aktivierte Achse (Servoversionen) 150 µsec. nominal (exakte Zeit ist 153.6 µsec.) pro aktivierte Achse (Schrittmotorversionen)
Endschalter	2 pro Achse: eine für jede Fahrtrichtung
Position Capture Triggers	2 pro Achse: Index und Referenzschalter
Andere digitale Signale (pro Achse)	1 AxisIn Signal pro Achse, 1 AxisOut Signal pro Achse
Software invertierbare Signale	Encoder A/A, Encoder B/B¯, Index I/I¯, Home, AxisIn, AxisOut, PositiveLimit, NegativeLimit (alle individuell programmierbar pro Achse)
Analogeingang	8 x 10-Bit analoge Eingänge
Trace Modi	einfach kontinuierlich
Max. Anzahl Tracevariablen	4
Anzahl verfügbarer Tracevariablen	27
Anzahl Kommandos	118-168 (abhängig von Motor- und Kartenversion)
I/Os	8 digitale Eingänge / TTL, active low 8 digitale Ausgänge / TTL, active low 8 analoge Eingänge
Spezielle Profilkombinationen	Trapezmodus mit Electronic Gearing S-Kurven Modus mit Electronic Gearing (eliminiert Überschießen) OPTION: Trapezmodus mit Electronic Gearing und Smoothing Faktor (eliminiert Überschießen und erlaubt on-the-fly-Änderung von Profilparametern)
Master/Slave Wechsel	Automatischer Master/Slave Wechsel möglich falls in Hostsoftware programmiert
Motor Check	programmierbare max. Schleppabstandsfehlererkennung mit oder ohne automatischer Motorabschaltung oder ignorieren
„On the fly“ Kontrolle	Von Profil und Filterparametern mit Pre-Load und individuellem oder simultanem Achsenupdate
Stromversorgung	4.80V to 5.25V; 1A

¤

Änderungen zu vorhergehenden Handbüchern

Changes to previous manuals

With the manual released on August 25 2004 we now start issuing all changes that affect the use of the POSYS® or software command structure. These changes can be found on this page in chronicle order of their change.

25.8.2004	ClearInterrupt	Argument AXIS added for use with POSYS® when used in CAN 2.0B mode.
25.8.2004	GetActualVelocity	Description more clearly explained
26.8.2004	GetInterruptAxis	Encoding changed
26.8.2004	GetPositionError	Addition to description
26.8.2004	GetVersion	Description changed
26.8.2004	ReadAnalog	Description corrected
26.8.2004	Reset	IMPORTANT NOTICE: Warning Note added
26.8.2004	Set/GetCommutationMode	Instance for mode added: microstepping; Description adjusted
30.08.2004	Set/GetIntegrationLimit	Returned data/Type corrected from signed 32 bits to unsigned 32 bits
30.08.2004	Set/GetSampleTime	values added for POSYS® 7xx, 8xx, 8xx-B and 9xx
30.08.2004	Set/GetSerialPortMode	Restrictions added for POSYS® 8xx-B and 9xx
30.08.2004	Set/GetSettleWindow	Values added for POSYS® 18xx/19xx
30.08.2004	Set/GetTracePeriod	values added for POSYS® 18xx/19xx
30.08.2004	Set/GetTraceStop	Examples added
28.09.2004	Analog Inputs 800-B version	The description showed that a second connector for analog inputs is available. This is not correct. The 800-B version ha only 1 possibility for connecting analog inputs. (J1 and J7). does not exist on the 800-B version card.
28.09.2004	Jumper description changed for POSYS® 800-B	With the exception of the jumper selection for the interrupt no jumpers are available on the 800-B card. This was described incorrectly in the "User settable components" section of the 800-B.
08.10.2004	Jumper description changed for POSYS® 1800	With the exception of the jumper selection for the interrupt no jumpers are available on the 1800 card. This was described incorrectly in the "User settable components" section of the 1800.
08.10.2004	Description added for POSYS® 1800	In the "User settable components" section the description fo " serial interface, CAN Interface, Motor Mode Jumpers and SYNC I/O were added and corrected.
29.10.2004	Description added for POSYS® 1800	In the "User settable components" section the descriptions for the connectors, card base address setting were added. The pinout for the extension connector and extended extension connector were added. The layout pictures with placement of switches, jumpers (interrupt selection) and connectors were added (front & ba side).
29.10.2004	Description added for POSYS® 1900	In the "User settable components" section the pinouts for th extension connector and extended extension connector wer added.
29.10.2004	Description added for POSYS® 900	In the "User settable components" section the pinout for the extension connector was added.
29.10.2004	Description added for POSYS® 800-B	In the "User settable components" section the pinout for the extension connector was added.
29.10.2004	Description added for POSYS® 800	In the "User settable components" section the pinout for the extension connector was added.
29.10.2004	Description added for POSYS® 700	In the "User settable components" section the pinout for the extension connector was added.
29.10.2004	Description changed for CAN interface (POSYS® 1800, 1900)	In the sections describing the usage with CAN the explanat was CANOpen. This was wrong and has been corrected to CAN 2.0B.

4.11.2004	Dimensions of POSYS® 1800	added
4.11.2004	Memory base address setting for POSYS® 1800 when equipped with dual port RAM	The dip switch (SW2) base address setting for the POSYS® 18xx when equipped with dual port RAM was added
04.11.2004	Analog pinout description to be added to the POSYS® 800-B, 900, 1800, 1900	The analog input pinout description for the POSYS® 800-B, 900, 1800 and 1900 has been corrected respectively added to the manual
24.11.2004	Safety instructions to be added	Safety Notice, Warranty and Disclaimer added
1.4.2005	Program description for the GUI version of the POSYS® test program for Windows	added
4.11.2004	Serial and CAN 2.0B	Setup and connection information added for POSYS® 8xx-B/18xx/9xx/19xx
27.1.2005	Changed jumper description for motor configuration (POSYS 18xx)	The jumper configuration was referring to the wrong jumpers. This was corrected. Instead of JP2 for stepper motor configuration it must state JP1 and instead of JP1 for servo and microstepping motors it must state JP2.
01.03.2005	Programmer's Info	Update and corrections on information about addressing differences of the POSYS® 7xx/8xx and 8xx-B
1.4.2005	Technical Note: setting of resistor networks for encoders	This section describes which pins are needed or not needed when using TTL incremental encoders or incremental encoders with inverted signals
15.4.2005	User settable components for POSYS® 700/800/800-B/ 1800	Exact information of placement of jumpers for IRQ setting added
10.5.2005	User settable components for POSYS® 1800	Descrition for usage of POSYS® 1800 with Dual Port RAM o the ISA-Bus added.
17.6.2005	Dual Port RAM Mode activation/deactivation	The dual Port RAM on the POSYS® 1800 needs to be activated or can be deactivated before access can take plac
17.6.2005	List of all POSYS® command	A list of all POSYS® commands is now also available right a the beginning of the Prgrammer's manual
5.7.2005: Change to Version 2.0. The version 2.0 of the manual and higher reflect the POSYS® software and hardware changes in regards to the driver (WinDriver) which is now the actual driver for Windows and Linux based operating systems.		
5.7.2005	Change of all POSYS® cards to the same driver (WinDriver)	All POSYS® motion controllers now use the same driver module designed for the bus (ISA or PC I) and operating system (Windows 98/2000/XP and Linux) they are used on.
5.7.2005	Change of card initialization	Due to the change of the driver the card initialization procedure changes, too. This effects all ISA-Bus based card which no longer need the command EnableIO. The comman OpenDevice as previously only needed for the PC I-based cards is now also the same (except parameters) for the ISA-based cards. For details please refer to the command OpenDevice/CloseDevice
5.7.2005	Commands added for access to Dual Port RAM	Following commands were added for easier access to Dual Port RAM (enable, disable, read and write)
26.7.2005	CommandSet/GetProfileKp added	The commandSet/GetProfileKpis an optional command to smooth trapezoidal accel./decel. motion down to a s-curve similar trajectory with the possibility to change all paramete like velocity, accel./decel., position "on-the-fly" which is not possible in pure s-curve profile mode.
17.08.2005	OpenDevice/CloseDevice	The english manual was not completed correctly in the version 2.1 from 27.07.2005
12.10.2005	Analog Inputs	Corrections made to definition (reference voltage) and pino positions of analog inputs. see various models for details.
28.10.2005	Sample Description of setting of DPRAM address for POSYS® 1800 corrected	The sample setting of the memory address setting for the DPRAM of the POSYS® 1800 was corrected. It was showing incorrectly the opposite setting for the address D0000. This effected only the english manual.
09.11.2005	Usage of On-Board Dual Port RAM with POSYS® 1800	Some information added for use of DPRAM with POSYS® 1800 PC/104 motion controllers
09.11.2005	Single Port Ram / Dual Port RAM	POSYS® GUI program for POSYS® 1800 translation finalize

07.04.2006	minor formatting adjustment	just some minor formatting adjustments for better printout with PDF and HTMLHELP.
11.04.2006	Linux Driver and Software Installation	Description for compiling and installing the software and drivers for PC I-Bus based POSYS® and PC 104-Bus based POSYS® added
25.7.2006	HardReset/MCReset	Failure in the description which was corrected
9.10.2006	Status Read Operation	Technical Note Addendum
17.01.2007	GetAxis_In_Out_Home	Argument description and Instance were corrected
17.01.2007	Description of the variable "CardHandle" for use with MotionScript changed	The description was corrected for usage with MotionScript® programming language
25.1.2007	Numbering of Breakpoint	The numbering of the value of the used Breakpoint in the Breakpoint examples for breakpointID differed from the val described in the command section. This was corrected.
30.1.2007	Activity Status Register	In the Activity Status Register the reserved Bit 6 and Axis-settled Bit 7 was not correctly defined for POSYS 700/800/800-B/900 in the User's Guide. The command description in the Programmer's manual allthough was defined correctly.
13.2.2007	Numbering of connection scheme changed for Option CON	The numbering scheme for the connections to the Option CON for the POSYS 800-B, 1800, 900 and 1900 series was changed for better regognition of correct placement.
27.4.2007	Added information to section OPTION CON	The numbering of the pinout for the 68-position connector f the POSYS 900 and 1900 series was confusing. Additional information was added to clarify the pinout description and numbering. Please read the information provided for OPTION CON AND 900/1900 which are different, depending on if you use the IO68 from POSYS Motion C ontrol or if you build yo own connector.
23.7.2007	Correction on SampleTime fo MC58120/MC55120	The SampleTime for the single axis/dual chip version of the Magellan motion control chip was incorrect. The correct SampleTime is 102.4 µs and NOT as indicated 51.2 µs. This also affects the POSYS® 18xx/19xx when equipped wi single axis/dual chip Magellan chip set OR if the POSYS® 18xx/19xx is used in such a way that the axes 2-4 have be deactivated.
24.7.2007	Set/GetBufferStart	Set/GetBufferStart had an incorrect BufferStart value for the POSYS® 1800/1900 boards indicating the same value as fo the POSYS® 700/800/800-B/900 boards. This has been corrected.
05.10.2007	InitializePOSYS	This command is only needed together with the old Kithara-Driver which is no longer supported. The information passed on with the command are all locate in the command OpenDevice
24.1.2008	Added connector description J12 for POSYS 1800 series	The description for the output signals of J12 for the POSYS 1800 series was added; depending on the type of motor an type of signal you wish to use this connector provides the possibility for connecting inverted PWM or Pulse & Direction signals.
27.2.2008	Jumper description corrected	On the page of the jumper description for correct motor output mode for the POSYS 1900 series the wrong jumper positions were defined. The opposite was correct. The description is now correct.
01.09.2008	Piezo (Ceramic) Motor Functions	This chapter was added to describe the custom feature on how to tune Piezo Motors being controlled like servo motors In addition on this page you will also find all commands related to this topic.
16.10.2008	Hardware & Treiber Installation: Rapid Driver	New drivers for POSYS® 800-B, 1800, 900 and 1900. This chapter describes the nstallation procedure
1.11.2008	Information added for Firmware-based >2.x POSYS boards 1800/1900	The POSYS 1800/1900 series are now available with Firmware version >2.x. With this firmware additional commands have been added or existing commands were expanded with additional parameters. The commands are: GetActiveMotorCommand (new) GetActiveOperatingMode (new) GetDriveStatus (new)

		GetInstructionError (same as GetHostIOError, but some parameters added) GetPositionLoopValue (new) HardReset (added variables to be resetted) ResetEventStatus (variable added) RestoreOperatingMode (new) Set/GetAxisOutMask (new) Set/GetBreakpoint (added variables) Set/GetBreakpointUpdateMask (new) Set/GetBreakpointValue (add variable) Set/GetEventAction (new) Set/GetHoldingCurrent (new) Set/GetOperatingMode (new) Set/GetPositionLoop (new) Set/GetSampleTime (different values for >2.x) Set/GetUpdateMask (new)
19.11.2008	SetEventAction Description added	SetEventAction was not yet added to the Usability Section of the manual. This was corrected
12.03.2009	Programmer's Info	The English manual did not contain complete information. This was corrected.
21.05.2009	External Profile Mode	The English section describing the usage of this mode was not translated. This is now done
08.12.2009	Set/GetEncoderSource	Added additional explanation for parameter loopback which is not available on POSYS® 85x-B and 95x series
05.02.2010	IO700/800-1	Description corrected regarding the jumper position for internal and external power supply
26.07.2010	GP CON Steckverbinder	Corrected the Tyco/AMP model numbers which can be used.
12.01.2012	Changed email addresses and company to POSYS Motion Control	Revision: 7.0
12.03.2012	Some corrections done regarding software packages and availability of functions	Support for circular interpolation only as option and not available for all models. Tuning software available using PMD's Pro-Motion software.

¤

Hardware

Hardware & Driver Installation

Hardware & Treiber Installation: WinDriver

Hardware und Treiber Installation

Neue Installationsprozedur nach Wechsel zu WinDriver. Bitte lesen. Falls sie einen früheren Treiber benötigen (Kithara) dann setzen Sie sich mit POSYS Motion Control GmbH & Co.KG in Verbindung.

Schalten Sie den Rechner aus. Installieren Sie auf keinen Fall PCI-Karten solange der Rechner eingeschaltet ist. Die Karten und der Rechner können Schaden nehmen. Die Karten werden definitiv vom System nicht erkannt. PCI-Karten müssen in einem PCI-Slot installiert werden, ISA-Karten in ISA-Slots. Die POSYS® 800/800B/1800 (PC-104 Bus) können in ISA-Slots installiert werden, falls eine Adapterkarte (PC/104 nach ISA) verwendet wird. Dies ist nur dann notwendig falls ein konventioneller PC oder IPC verwendet wird.

PCI Bus

Windows

Nachdem der Computer ausgeschaltet wurde, können Sie die POSYS® 900/1900 Karte(n) in einen freien PCI-Slot des Computers stecken. Sichern Sie die Karte am Slot und schalten Sie den Rechner wieder ein. Sobald das Betriebssystem (Windows 95/98/ME, NT/2000/XP) die neue Hardware entdeckt (pnp) hat, werden Sie aufgefordert die Treiber zu installieren. Teilen Sie dem Betriebssystem den Ort mit (/WD Driver Setup). Alle erforderlichen Dateien werden in die korrekten Verzeichnisse kopiert.

Falls die Installation erfolgreich war, können die installierten Karten zur Überprüfung verifiziert werden indem die Systemsteuerung und dort der Gerätemanager aufgerufen wird. Erfolgreich installierte Karten werden unter "Multifunktions Adapter" gelistet.

README.TXT	Wichtige Installationsinformationen für die Installation unter verschiedenen Windows Betriebssystemen. BITTE LESEN!!!
POSYS900WD.INF	für die POSYS® 900 Serie
POSYS1900WD.INF	für die POSYS® 1900 Serie
WINDRVR6.SYS	Treiber
WINDRVR6.INF	Inf-Datei
WDREG16.EXE	für Windows 95/98/ME
WDREG.EXE	für Windows NT/2000/XP
WDREG_GUI.EXE	für Windows (alle Versionen)
WINDRVR6.VXD	für Windows 95/98/ME
WD_UTILS.DLL	für Windows (alle Versionen)
1900_XP.BAT	für Windows (alle Versionen)

Falls Sie wünschen das Beispielprogramm POSYS900V4.EXE oder POSYS1900V4.EXE zu verwenden, ist es notwendig bestimmte OCX-Dateien im Betriebssystem an zu melden. Wechseln Sie in das Unterverzeichnis /WD Driver Setup/OCX der Installations CD und rufen Sie die Batch-Datei "registerOCX.bat" auf. Abhängig vom verwendeten Betriebssystem kann es sein, dass Sie zur Bestätigung zweier Fenster aufgerufen werden.

Linux

Für die PCI-Karten werden seperat für Linux 2 ZIP-Dateien geliefert die wiederum in einer ZIP-Datei zusammengefasst sind. Die Datei "REDIST.ZIP" enthält alle erforderlichen Dateien für die Treiberinstallation. Die Datei "DISTRIBUTE.ZIP" enthält den Quellcode für die POSYS 900/1900 Karten, inklusive Beispielprogramme für die Konsole die sofort verwendet werden können um die Kommunikation mit den Karten zu testen. Wichtig ist auch, dass die Kernel-Soures und der GNU C/C++-Compiler installiert sein müssen.

Treiberinstallation:

- Es ist notwendig als *root* oder mit *su passwd* im Terminalfenster angemeldet zu sein.
- Entpacken Sie die Verzeichnisse vorzugsweise nach */usr/local*. Dort wird ein neues Verzeichnis erstellt. Dies ist das Verzeichnis */WinDriver*.
- Wechseln Sie in das Verzeichnis */WinDriver/redist* und prüfen Sie ob die Dateien *configure*, *setup_inst_dir* und *wdreg* als ausführbare Dateien angelegt wurden. Falls nicht dann führen Sie für jede Datei folgendes Kommando aus: *chmod datei 755*
- Führen Sie anschliessend *sh ./configure* oder einfach *./configure* aus. Das Makefile wird erstellt.
- Führen Sie *make* aus und anschliessend *make install*
- Der Treiber sollte anschliessend geladen worden sein. Das können Sie durch Aufruf von *lsmod* überprüfen. Unter den aufgeführten Modulen sollte jetzt auch das Modul *windrvr6* aufgeführt werden.
- Die Treiberinstallation ist erfolgreich beendet, nun kann Kommunikation mit der PC I-Karte erfolgen.

- Im Boot-Script sollten Sie noch einen Eintrag machen der dafür sorgt, dass der Treiber beim Booten des Rechners automatisch geladen wird.

Softwareinstallation:

- Entpacken Sie die ZIP-Datei *Distribute* in das Unterverzeichnis */usr/local/POSYS*
- Im Unterverzeichnis von */POSYS* werden weitere Verzeichnisse angelegt. Diese sind:
  /source
  /test1900
  /test900
  /include
  /object
  /module
- Im Verzeichnis */source* sind alle Dateien mit Quellcode abgelegt, die für die Kommunikation und den Betrieb mit der POSYS® PCI-Karte benötigt werden. Diese können für die Erstellung eigener Applikationen verwendet und eingebunden werden. Die Treiber-Dateien können aus lizenz-rechtichen Gründen nicht im Quellcode mitgeliefert werden.
- Im Verzeichnis */test1900* ist ein vorbereitetes Linux Konsolenprogramm. Nach Aufruf von *make* und *make install* wird die für Ihr System gültige Version kompiliert. Sie finden anschliessend die Datei *Test1900* im Verzeichnis vor. Durch Eingabe von ./Test1900 können sie das Programm starten.
- Im Verzeichnis */test900* ist ein vorbereitetes Linux Konsolenprogramm. Nach Aufruf von *make* und *make install* wird die für Ihr System gültige Version kompiliert. Sie finden anschliessend die Datei *Test900* im Verzeichnis vor. Durch Eingabe von ./Test900 können sie das Programm starten.
- Im Verzeichnis */include* sind alle Include-Dateien abgelegt.
- Im Verzeichnis */object* finden Sie die dazugehörigen Objekt-Dateien

ISA/PC-104 Bus

Windows

Wechseln Sie in das Verzeichnis /WD Driver Setup und rufen Sie das Programm WDREG_GUI.EXE auf wie für das verwendete Betriebssystem erforderlich (näheres in der README.TXT). Dies führt zur Installation aller erforderlichen Treiberdateien und Treiber-DLLs für Windows 95/98/ME oder NT/2000/XP. Es wird wahrscheinlich erforderlich sein anschliessend den Rechner neu zu starten nach der Treiberinstallation. Für den ISA-Bus gibt es keine Überprüfung auf Adressenkonflikte.

Als nächsten Schritt können Sie das Utility-Programm POSYSDriverXSetup aufrufen. Mit diesem Programm können Sie bis zu 16 Karten in der Registry eintragen inkl. Adresse und evtl. verwendetem Interrupt.

README.TXT	Wichtige Installationsinformationen für die Installation unter verschiedenen Windows Betriebssystemen. BITTE LESEN!!!
WINDRVR6.SYS	Treiber
WINDRVR6.INF	Inf-Datei
WDREG.EXE	für Windows 95/98/ME
WDREG16.EXE	für Windows NT/2000/XP
WDREG_GUI.EXE	für Windows (alle Versionen)
WINDRVR6.VXD	für Windows 95/98/ME
WD_UTILS.DLL	für Windows (alle Versionen)

Falls Sie wünschen das Beispielprogramm POSYS1800V4.EXE/POSYS800BV4.EXE oder POSYS700V4.EXE zu verwenden, ist es notwendig bestimmte OCX-Dateien im Betriebssystem an zu melden. Wechseln Sie in das Unterverzeichnis /WD Driver Setup/OCX der Installations CD und rufen Sie die Batch-Datei "registerOCX.bat" auf. Abhängig vom verwendeten Betriebssystem kann es sein, dass Sie zur Bestätigung zweier Fenster aufgerufen werden.

Linux

Für die POSYS® PC104-Karten werden seperat für Linux 2 ZIP-Dateien geliefert die wiederum in einer ZIP-Datei zusammengefasst sind. Die Datei "REDIST.ZIP" enthält alle erforderlichen Dateien für die Treiberinstallation. Die Datei "DISTRIBUTE.ZIP" enthält den Quellcode für die POSYS 800-B/1800 Karten, inklusive Beispielprogramme für die Konsole die sofort verwendet werden können um die Kommunikation mit den Karten zu testen. Wichtig ist auch, dass die Kernel-Soures und der GNU C/C++-Compiler installiert sein müssen.

Treiberinstallation:

- Es ist notwendig als *root* oder mit *su passwd* im Terminalfenster angemeldet zu sein.
- Entpacken Sie die Verzeichnisse vorzugsweise nach */usr/local*. Dort wird ein neues Verzeichnis erstellt. Dies ist das

Verzeichnis */WinDriver*.

- Wechseln Sie in das Verzeichnis */WinDriver/redist* und prüfen Sie ob die Dateien *configure*, *setup_inst_dir* und *wdreg* als ausführbare Dateien angelegt wurden. Falls nicht dann führen Sie für jede Datei folgendes Kommando aus: *chmod datei 755*
- Führen Sie anschliessend *sh ./configure* oder einfach *./configure* aus. Das Makefile wird erstellt.
- Führen Sie *make* aus und anschliessend *make install*
- Der Treiber sollte anschliessend geladen worden sein. Das können Sie durch Aufruf von *lsmod* überprüfen. Unter den aufgeführten Modulen sollte jetzt auch das Modul *windrvr6* aufgeführt werden.
- Die Treiberinstallation ist erfolgreich beendet, nun kann Kommunikation mit der PC 104-Karte erfolgen.
- Im Boot-Script sollten Sie noch einen Eintrag machen der dafür sorgt, dass der Treiber beim Booten des Rechners automatisch geladen wird.

Softwareinstallation:

Die Posys1800 ist ein Konsolenprogramm für Linux-Distributionen. Es kann keine Garantie übernommen werden, dass es auf allen Linux Distributionen funktioniert. Es wurde auf SuSE Linux 10.0 getestet. Die Software setzt voraus, dass die Treiberinstallation erfolgreich war und der Treiber auch geladen ist.

Das Testprogramm stellt alle Datien auch im Quellcode zur Verfügung, welche der Entwickler für seine Applikation anpassen kann.

Falls die POSYS® noch nicht installiert wurde, dann sollte dies jetzt erfolgen. Stellen Sie die Karte auf eine freie Basisadresse ein. Bei Auslieferung ist die Karte standardmäßig auf 0x300 (768 dezimal) eingestellt. Ist eine andere als die eingestellte Adresse notwendig, verweisen wir an dieser Stelle an die Hardwareeinstellungen der jeweiligen Karten in der Sektion POSYS® Versionsspezifikationen. Für die POSYS® 1800 gilt zudem, dass ausgewählt werden kann ob und mit welcher Adresse der standardmäßige DPRAM angesprochen werden kann. Hierzu verweisen wir auf die Sektion Verwendung des On-Board Dual Port RAMs mit der POSYS® 1800. Die mitgelieferte Software ist darauf eingestellt mit maximal 3 Karten eine Verbindung aufbauen zu können. Dies kann aber für die Verwendung von mehr Karten ohne weiteres erweitert werden. Nachdem alle Vorarbeiten erledigt sind kann die Software installiert werden:

- Falls beim Booten das windrvr6 Modul nicht geladen wurde, dann holen Sie das jetzt bitte im Verzeichnis /usr/local/WinDriver/redist mit dem Kommando *./wdreg windrvr6. no* nach. Geben Sie bitte auch das "no" ein.
- Nun wechseln Sie in das Verzeichnis /usr/local/posys_pc104/linux/Release_Build und geben Sie das Kommando ./Posys1800 ein. Dies startet das Konsolenprogramm und Sie werden nach gewissen Eingaben gefragt.
- Geben Sie ein wieviele Karten installiert sind (1 oder 2).
- Geben Sie die Basis IO Adresse ein als Hex-Nummer. Es muss mit der eingestellten Adresse auf der Karte übereinstimmen.
- Falls eine POSYS 1800 installiert wurde, dann können Sie jetzt auch die Adresse des eingestellten Dual Port RAMS eingeben. Falls es sich um eine POSYS 800-B handelt, kann dieser Schritt übersprungen werden.
- Falls ein Interrupt verwendet werden soll, dann geben Sie hier die Interruptnummer ein. Auch dies muss mit dem eingestellten Interrupt auf der Karte übereinstimmen. Dafür muss der entsprechende Jumper gesetzt worden sein.
- Nachdem all diese Daten übergeben worden sind, startet der Treiber und die Karte wird initialisiert. Alle POSYS Funktion können nun ausgeführt werden. Falls ein Fehler auftrat, wird ein Nachricht ausgegeben und das Programm beendet.

Das Programm wurde ursprünglich mit Borland's C ++BuilderX IDE für Linux entwickelt. Es wurde mit dem GNU C ompiler kompiliert. Ein *makefile* wird auch mitgeliefert und befindet sich im Verzeichnis */usr/local/posys_pc104*. Für die Rekompilierung muss der GNU C ompiler auf dem Zielsystem installiert sein.

DOS

Für DOS ist keine Treiberinstallation notwendig. IO und Speicher können direkt aus den Programmiersprachen C, Pascal und Basic adressiert werden.

¤

Hardware & Treiber Installation: Rapid Driver

Hardware und Treiber Installation

Neue Installationsprozedur nach Wechsel zu Rapid Driver. Bitte lesen. Falls sie einen früheren Treiber benötigen (Kithara, WinDriver) dann setzen Sie sich mit POSYS Motion Control GmbH & Co.KG in Verbindung.

Schalten Sie den Rechner aus. Installieren Sie auf keinen Fall PCI-Karten solange der Rechner eingeschaltet ist. Die Karten und der Rechner können Schaden nehmen. Die Karten werden definitiv vom System nicht erkannt. PCI-Karten müssen in einem PCI-Slot installiert werden, ISA-Karten in ISA-Slots. Die POSYS® 800/800B/1800 (PC-104 Bus) können in ISA-Slots installiert werden, falls eine Adapterkarte (PC/104 nach ISA) verwendet wird. Dies ist nur dann notwendig falls ein konventioneller PC oder IPC verwendet wird.

PCI Bus

Windows

Nachdem der Computer ausgeschaltet wurde, können Sie die POSYS® 900/1900 Karte(n) in einen freien PCI-Slot des Computers stecken. Sichern Sie die Karte am Slot und schalten Sie den Rechner wieder ein. Sobald das Betriebssystem (Windows 95/98/ME, NT/2000/XP) die neue Hardware entdeckt (pnp) hat, werden Sie aufgefordert, die Treiber zu installieren. Teilen Sie dem Betriebssystem den Ort mit (/RD Installlation). Alle erforderlichen Dateien werden in die korrekten Verzeichnisse kopiert.

Nach erfolgreicher Installation, können die installierten Karten zur Überprüfung verifiziert werden, indem die Systemsteuerung und dort der Gerätemanager aufgerufen wird. Erfolgreich installierte Karten werden unter "Multifunktions Adapter" gelistet.

POSYS900WD.INF	für die POSYS® 900 Serie
POSYS1900WD.INF	für die POSYS® 1900 Serie
RapidIsa.SYS	Treiber
POSYS900.INF	Inf-Datei für POSYS 900
POSYS1900.INF	Inf-Datei für POSYS 1900
RdInstall98.EXE	für Windows 95/98/ME
RdInstall.EXE	für Windows 2000/XP
RdInst16.DLL	Programmbibliothek
RapidInstaller32.dll	Programmbibliothek
RapidIsa.dll	Programmbibliothek
ReadMe.rtf / ReadMe.txt	Installationsanweisung

Falls Sie wünschen das Beispielprogramm POSYS900V4.EXE oder POSYS1900V4.EXE zu verwenden, ist es notwendig bestimmte OCX-Dateien im Betriebssystem anzumelden. Wechseln Sie in das Unterverzeichnis /OCX der Installations CD und rufen Sie die Batch-Datei "registerOCX.bat" auf. Abhängig vom verwendeten Betriebssystem kann es sein, dass Sie zur Bestätigung zweier Fenster aufgerufen werden.

Linux (benutzen Sie bitte den WinDriver)

ISA/PC-104 Bus

Windows

Wechseln Sie in das Verzeichnis /WD Driver Setup und rufen Sie das Programm WDREG_GUI.EXE auf wie für das verwendete Betriebssystem erforderlich (näheres in der README.TXT). Dies führt zur Installation aller erforderlichen Treiberdateien und Treiber-DLLs für Windows 95/98/ME oder NT/2000/XP. Es wird wahrscheinlich erforderlich sein anschliessend den Rechner neu zu starten nach der Treiberinstallation. Für den ISA-Bus gibt es keine Überprüfung auf Adressenkonflikte.

Als nächsten Schritt können Sie das Utility-Programm POSYSDriverXSetup aufrufen. Mit diesem Programm können Sie bis zu 16 Karten in der Registry eintragen inkl. Adresse und evtl. verwendetem Interrupt.

RapidIsa.SYS	Treiber
POSYS800.INF	Inf-Datei für POSYS 800-B
POSYS1800.INF	Inf-Datei für POSYS 1800
RdInstall98.EXE	für Windows 95/98/ME
RdInstall.EXE	für Windows 2000/XP
RdInst16.DLL	Programmbibliothek
RapidInstaller32.dll	Programmbibliothek
RapidIsa.dll	Programmbibliothek

ReadMe.rtf / ReadMe.txt Installationsanweisung

Falls Sie wünschen das Beispielprogramm POSYS1800V4.EXE/POSYS800BV4.EXE oder POSYS700V4.EXE zu verwenden, ist es notwendig bestimmte OCX-Dateien im Betriebssystem anzumelden. Wechseln Sie in das Unterverzeichnis /OCX der Installations CD und rufen Sie die Batch-Datei "registerOCX.bat" auf. Abhängig vom verwendeten Betriebssystem kann es sein, dass Sie zur Bestätigung zweier Fenster aufgerufen werden.

Linux (benutzen Sie bitte den WinDriver)

¤

POSYS® Versionsspezifikationen

POSYS® 8xx-B

Versionen 8xx-B

Liste der verfügbaren Versionen

POSYS® 80x-B für bürstenbehaftete Servomotoren

Die POSYS® 80x-B kann auch mit bürstenlosen Servomotoren eingesetzt werden vorausgesetzt entsprechende Verstärker werden verwendet.

Modell	# Achsen
POSYS® 802-B	2
POSYS® 804-B	4

POSYS® 85x-B für Schrittmotoren

Die POSYS® 85x-B kann auch für Mikroschrittmotoren verwendet werden wenn Mikroschritttreiber verwendet werden.

Modell	# Achsen
POSYS® 852-B	2
POSYS® 854-B	4

Alle Versionen sind auch für den erweiterten Temperaturbereich erhältlich (-40° - +85°C). Nicht auf Lager. Für Bestellungen fügen sie der Bestellnummer ein "-I" hinzu (z.B.: 804-BI).

¤

Einstellbare Komponenten

Einstellbare Komponenten; POSYS® 8xx-B

In diesem Abschnitt erhalten Sie die Informationen die notwendig sind alle Komponenten korrekt mit einander zu verbinden. Das folgende Bild zeigt die Position der einzelnen Verbindungen auf mit ihren Bezeichnungen zur leichteren Orientierung.

Die wichtigsten anwender bezogenen Komponenten sind:

- Widerstandsnetzwerke RS1, RS2 und RS3. Sie bestimmen die Art der Enkoderterminierung
- Mode Jumper erlaubt die serielle Schnittstelle als primärer Kommunikationsport ausgewählt werden zu können. Ansonsten (Standard) ist es der PC-Bus. Ebenso erlaubt die POSYS 8xx-B diese Schnittstelle als Kommunikationsport zu verwenden. Die POSYS 7xx und 8xx verfügen nicht über eine serielle Schnittstelle
- Sync I/O um die Karte in den Master oder Slave Modus zu schalten, falls weitere Karten im System eingesetzt werden.

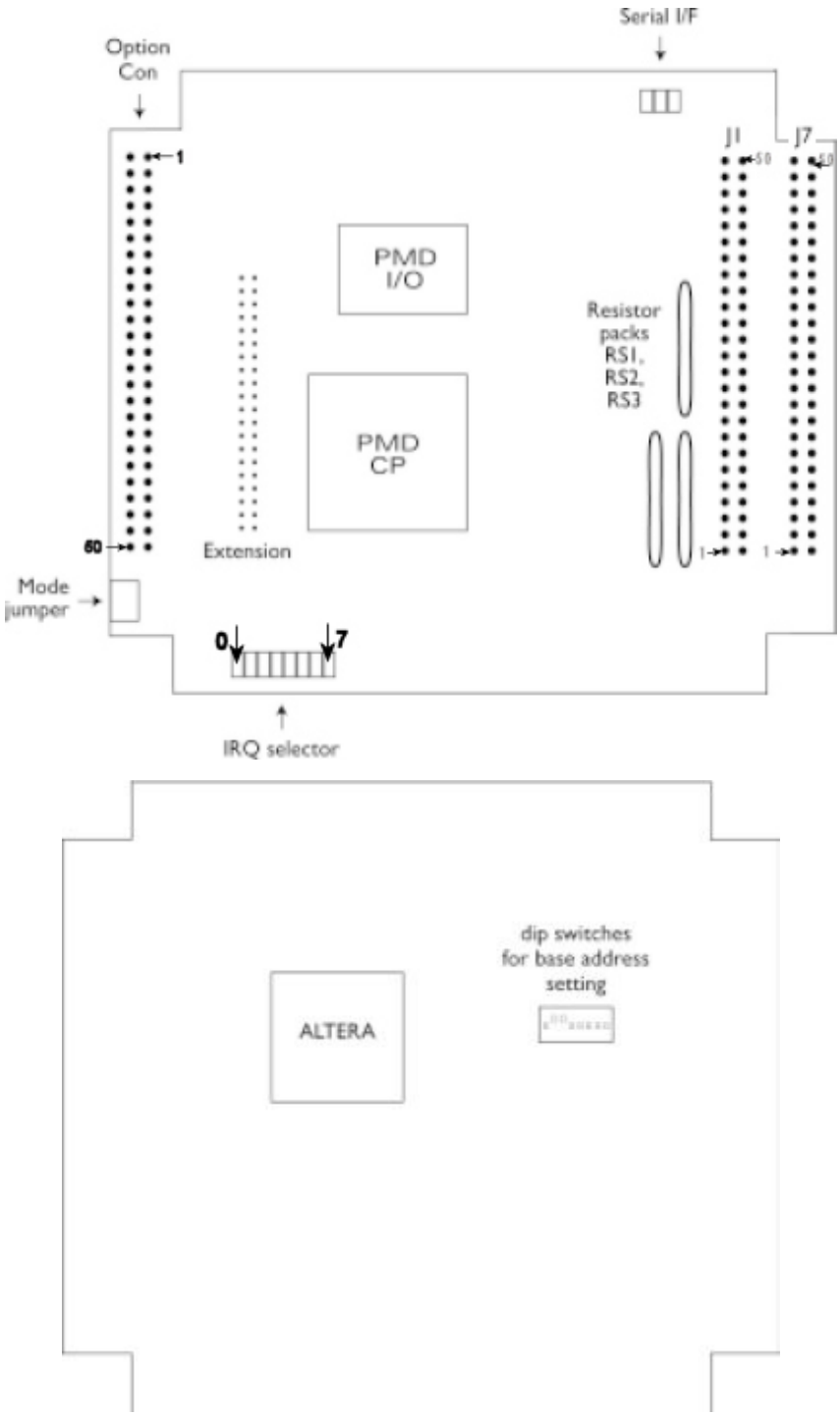


Abbildung zeigt POSYS 8xx-B

Widerstandsnetzwerke

Die folgende Tabelle zeigt die Funktionen der Widerstandsnetzwerke RS1-RS3 auf:

Komponente	Stellung	Beschreibung
Netzwerk-widerstände	Installiert: RS1-RS3 (Standard)	Für die Verwendung von Enkodern mit invertierten Signalen sollten die Widerstandsnetzwerke installiert bleiben
	Deinstalliert: RS1-RS3	Für Enkoder mit TTL-Signalausgabe
	Sollten beide Versionen von Enkodern im System eingesetzt werden, ist es auch möglich die Widerstandsnetzwerke entsprechend anzupassen. Bitte setzen Sie sich mit uns in Verbindung für die genauen Pinpositionen der Widerstandsnetzwerke	

Falls sowohl Enkoder mit TTL als auch Enkoder mit invertierten Signalen Verwendung finden, können Sie auf der Seite Resistor Networks for TTL or inverted Encoder Signals Detailinformationen finden wie die Widerstandsnetzwerke einzustellen sind.

Dipschalter und Jumper Stellungen

Die POSYS 7xx/8xx und 8xx-B haben Dipschalter für Adressinformationen und Steckbrücken um bestimmte Funktionalitäten zu de- oder aktivieren. Diese Stellungen sind in der folgenden Tabelle näher beschrieben:

Item	Einstellung	Beschreibung
SW1	S1-1 8 (hex) S1-2 10 (hex) S1-3 20 (hex) S1-4 40 (hex) S1-5 80 (hex) S1-6 100 (hex) S1-7 200 (hex) S1-8 400 (hex) 300 (hex) ist die Standardeinstellung	Switch S1 setzt die Kartenadresse für den ISA-Bus. Die gewählte Adresse ist der kumulierte Wert aus den entsprechenden Einstellungen wie sie links beschrieben sind. Ist der Schalter in der AUS-Position wird der Wert zum endgültigen Wert hinzu gerechnet. z.B um den Wert 340 (hex) einzustellen würden die Schalter auf folgenden Stellungen gesetzt: S1-1 on S1-2 on S1-3 on S1-4 off S1-5 on S1-6 off S1-7 off S1-8 on Die Kartenadresse eingestellt auf der Karte muss genau der Adresse in Ihrer Software entsprechen. Ausserdem darf diese Adresse von keiner anderen Karte belegt sein. Ausgehend von der Basisadresse belegt die POSYS anschliessend 8 Adressen. z.B. falls 300 (hex) eingestellt wurde, werden die Adressen von 300-307 (hex) belegt.
Jumper 17	Nicht gesteckt (standard)	Mit den Steckbrücken #17 kann der gewünschte zu verwendende Interrupt selektiert werden. Nur ein Interrupt kann ausgewählt werden und dieser Interrupt darf auch sonst nicht von anderen Komponenten bereits belegt sein. Um z.B. IRQ3 auszuwählen muss die Steckbrücke 0 gesteckt sein. J17 Designation / IRQ Selector — ISA IRQ No. 0 — 3 1 — 5 2 — 7 3 — 10 4 — 11 5 — 12 6 — 14 7 — 15

Mode Jumper

Die folgende Tabelle zeigt die Funktionen des Mode Jumpers auf. Die Jumper ist nur auf den Karten POSYS 8xx-B und POSYS 9xx vorhanden. Die POSYS 7xx und 8xx verfügen nicht über diesen Jumper:

Komponente	Stellung	Beschreibung
Mode Jumper	1-2 (Standard)	Der PC-Bus ist in dieser Stellung als Hauptkommunikationsleitung definiert
	2-3	Der PC-Bus als Kommunikationsleitung ist deaktiviert, es ist nur noch möglich über die serielle Schnittstelle (RS232 = Standard; RS422/485 über optionale Module zusätzlich möglich) zu kommunizieren.

¤

Serielle Schnittstelle

Serial I/F Stecker PC/104 Karte (POSYS 8xx-B)

Nur gültig für die POSYS 8xx-B Serie

Dieser Stecker stellt eine serielle Verbindung zur POSYS 8xx-B zur Verfügung. Der Serial I/F Stecker ist ein 6-poliger Molex MLX Micro-4-SMD Stecker

Pin	Verbindung	Beschreibung
1	SrlEnable	Serial enable (wird nur für RS422/485 benötigt)
2	SrlXmt	Serial transmit output
3	SrlRcv	Serial receive input
4	GND	Ground
5	Vcc	+5V
6	n.c.	No connection

Diese Schnittstelle erlaubt die POSYS 8xx-B über die serielle Schnittstelle RS232 zu betreiben. Sie kann als primärer Kommunikationsschnittstelle eingestellt werden und erlaubt die Verwendung aller Kommandos oder als sekundäre Schnittstelle über die nut GET-Kommandos für Überwachungszwecke akzeptiert werden.

Das serielle Kabel ist optional. Die folgende Beschreibung listet die notwendigen Komponenten auf falls es gewünscht ist ein eigenes serielles Kabel zu verwenden. Der Stecker für die serielle Verbindung ist ein 6-poliger Molex MLX Micro-4-SMD Typ Stecker.

Die Standardeinstellungen für die serielle Kommunikation nach dem Einschalten sind:
Geschwindigkeit: 9600 Baud/sek.
Parität: keine
Stop Bit: 1
Protokoll: point-to-point

Falls andere Standardeinstellungen gewünscht werden, dann setzen Sie sich bitte mit POSYS Motion Control GmbH & Co.KG in Verbindung. Diese Werte sind aus kompatibilitätsgründen gewählt worden, können aber kundenspezifisch angepasst werden. Beim Einschalten würden diese neuen Werte sofort effektiv.

Set/GetSerialPortMode, Serial Port Description

¤

Bürstenbehaftete Servomotor Pinout Beschreibung

Pinout Beschreibung für bürstenbehaftete oder bürstenlose (mit externe Kommutierung) Servomotoren

POSYS® 80x-B
Pin 1-50 = J7, Pin 51-100 = J1

Verbindungen auf der Karte:
TSM-125-01-T-DV (Samtec)
Gegenstecker: IDSD-25-S

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
1	A1+	Channel A+ encoder signal (X)	51	A3+	Channel A+ encoder signal (Z)
2	A1-	Channel A- inverted encoder signal (X)	52	A3-	Channel A- inverted encoder signal (Z)
3	B1+	Channel B+ encoder signal (X)	53	B3+	Channel B+ encoder signal (Z)
4	B1-	Channel B- inverted encoder signal (X)	54	B3-	Channel B- inverted encoder signal (Z)
5	I1+	Channel I+ encoder signal (X)	55	I3+	Channel I+ encoder signal (Z)
6	I1-	Channel I- inverted encoder signal (X)	56	I3-	Channel I- inverted encoder signal (Z)

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
7	Vc1	+5V	57	Vc1	+5V
8	GND	Ground	58	GND	Ground
9	XPLIM1	Limit switch positive direction (X)	59	XPLIM3	Limit switch positive direction (Z)
10	XNLIM1	Limit switch negative direction (X)	60	XNLIM3	Limit switch negative direction (Z)
11	HOME1	Home signal (X) or high speed latch	61	HOME3	Home signal (Z) or high speed latch
12	GND	Ground	62	GND	Ground
13	OUT1	User programmable output (X)	63	OUT3	User programmable output (Z)
14	BPWM1	PWM magnitude (X)	64	BPWM3	PWM magnitude (Z)
15	BPWMS1	PWM sign (X)	65	BPWMS3	PWM sign (Z)
16	AXIN1	User programmable input (X)	66	AXIN3	User programmable input (Z)
17	DACV1	Motor command (X), ±10V signal	67	DACV3	Motor command (Z), ±10V signal
18	AGND	Ground for motor command	68	AGND	Ground for motor command
19	A2+	Channel A+ encoder signal (Y)	69	A4+	Channel A+ encoder signal (W)
20	A2-	Channel A- inverted encoder signal (Y)	70	A4-	Channel A- inverted encoder signal (W)
21	B2+	Channel B+ encoder signal (Y)	71	B4+	Channel B+ encoder signal (W)
22	B2-	Channel B- inverted encoder signal (Y)	72	B4-	Channel B- inverted encoder signal (W)
23	I2+	Channel I+ encoder signal (Y)	73	I4+	Channel I+ encoder signal (W)
24	I2-	Channel I- inverted encoder signal (Y)	74	I4-	Channel I- inverted encoder signal (W)
25	Vc1	+5V	75	Vc1	+5V
26	GND	Ground	76	GND	Ground
27	XPLIM2	Limit switch positive direction (Y)	77	XPLIM4	Limit switch positive direction (W)
28	XNLIM2	Limit switch negative direction (Y)	78	XNLIM4	Limit switch negative direction (W)
29	HOME2	Home signal (Y) or high speed latch	79	HOME4	Home signal (W) or high speed latch
30	OUT2	User programmable output (Y)	80	OUT4	User programmable output (W)
31	BPWM2	PWM magnitude (Y)	81	BPWM4	PWM magnitude (W)
32	BPWMS2	PWM sign (Y)	82	BPWMS4	PWM sign (W)
33	AXIN2	User programmable input (Y)	83	AXIN4	User programmable input (W)
34	DACV2	Motor command (Y), ±10V signal	84	DACV4	Motor command (W), ±10V signal
35	AGND	Ground for motor command	85	AGND	Ground for motor command
36	UI0	uncommitted input (0)	86	UI4	uncommitted input (4)
37	UI1	uncommitted input (1)	87	UI5	uncommitted input (5)
38	UI2	uncommitted input (2)	88	UI6	uncommitted input (6)
39	UI3	uncommitted input (3)	89	UI7	uncommitted input (7)
40	AMPEN1	Amplifier enable (X) or user programmable output	90	AMPEN3	Amplifier enable (Z) or user programmable output
41	UO0	uncommitted output (0)	91	UO4	uncommitted output (4)
42	UO1	uncommitted output (1)	92	UO5	uncommitted output (5)
43	UO2	uncommitted output (2)	93	UO6	uncommitted output (6)
44	UO3	uncommitted output (3)	94	UO7	uncommitted output (7)
45	AMPEN2	Amplifier enable (Y) or user programmable output	95	AMPEN4	Amplifier enable (W) or user programmable output
46	RS-	Hardware reset	96	ANGND	GND for analog inputs
47	Ainput1	analog input channel (0)	97	Ainput5	analog input channel (4)

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
48	Ainput2	analog input channel (1)	98	Ainput6	analog input channel (5)
49	Ainput3	analog input channel (2)	99	Ainput7	analog input channel (6)
50	Ainput4	analog input channel (3)	100	Ainput8	analog input channel (7)

¤

Schrittmotor Pinout Beschreibung

Pinout Beschreibung Schritt-/Mikroschrittmotor* Version

POSYS® 85x-B
Pin 1-50 = J7, Pin 51-100 = J1

Verbindungen auf der Karte:
TSM-125-01-T-DV (Samtec)
Gegenstecker: IDSD-25-S

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
1	A1+	Channel A+ encoder signal (X)	51	A3+	Channel A+ encoder signal (Z)
2	A1-	Channel A- inverted encoder signal (X)	52	A3-	Channel A- inverted encoder signal (Z)
3	B1+	Channel B+ encoder signal (X)	53	B3+	Channel B+ encoder signal (Z)
4	B1-	Channel B- inverted encoder signal (X)	54	B3-	Channel B- inverted encoder signal (Z)
5	I1+	Channel I+ encoder signal (X)	55	I3+	Channel I+ encoder signal (Z)
6	I1-	Channel I- inverted encoder signal (X)	56	I3-	Channel I- inverted encoder signal (Z)
7	Vc1	+5V	57	Vc1	+5V
8	GND	Ground	58	GND	Ground
9	XPLIM1	Limit switch positive direction (X)	59	XPLIM3	Limit switch positive direction (Z)
10	XNLIM1	Limit switch negative direction (X)	60	XNLIM3	Limit switch negative direction (Z)
11	HOME1	Home signal (X) or high speed latch	61	HOME3	Home signal (Z) or high speed latch
12	GND	Ground	62	GND	Ground
13	OUT1	User programmable output (X)	63	OUT3	User programmable output (Z)
14	PULSE1	Pulse (X)	64	PULSE3	Pulse (Z)
15	DIR1	Direction (X)	65	DIR3	Direction (Z)
16	AXIN1	User programmable input (X)	66	AXIN3	User programmable input (Z)
17	ATREST1	At Rest (X)	67	ATREST3	At Rest (Z)
18	GND	Ground	68	GND	Ground
19	A2+	Channel A+ encoder signal (Y)	69	A4+	Channel A+ encoder signal (W)
20	A2-	Channel A- inverted encoder signal (Y)	70	A4-	Channel A- inverted encoder signal (W)
21	B2+	Channel B+ encoder signal (Y)	71	B4+	Channel B+ encoder signal (W)
22	B2-	Channel B- inverted encoder signal (Y)	72	B4-	Channel B- inverted encoder signal (W)
23	I2+	Channel I+ encoder signal (Y)	73	I4+	Channel I+ encoder signal (W)
24	I2-	Channel I- inverted encoder signal (Y)	74	I4-	Channel I- inverted encoder signal (W)
25	Vc1	+5V	75	Vc1	+5V
26	GND	Ground	76	GND	Ground
27	XPLIM2	Limit switch positive direction (Y)	77	XPLIM4	Limit switch positive direction (W)

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
28	XNLIM2	Limit switch negative direction (Y)	78	XNLIM4	Limit switch negative direction (W)
29	HOME2	Home signal (Y) or high speed latch	79	HOME4	Home signal (W) or high speed latch
30	OUT2	User programmable output (Y)	80	OUT4	User programmable output (W)
31	PULSE2	Pulse (Y)	81	PULSE4	Pulse (W)
32	DIR2	Direction (Y)	82	DIR4	Direction (W)
33	AXIN2	User programmable input (Y)	83	AXIN4	User programmable input (W)
34	ATREST2	At Rest (Y)	84	ATREST4	At Rest (W)
35	GND	Ground	85	GND	Ground
36	UI0	uncommitted input (0)	86	UI4	uncommitted input (4)
37	UI1	uncommitted input (1)	87	UI5	uncommitted input (5)
38	UI2	uncommitted input (2)	88	UI6	uncommitted input (6)
39	UI3	uncommitted input (3)	89	UI7	uncommitted input (7)
40	ATREST1	At Rest (X)	90	ATREST3	At Rest (Z)
41	UO0	uncommitted output (0)	91	UO4	uncommitted output (4)
42	UO1	uncommitted output (1)	92	UO5	uncommitted output (5)
43	UO2	uncommitted output (2)	93	UO6	uncommitted output (6)
44	UO3	uncommitted output (3)	94	UO7	uncommitted output (7)
45	ATREST2	At Rest (Y)	95	ATREST4	At Rest (W)
46	RS-	Hardware reset	96	ANGND	GND for analog inputs
47	Ainput1	analog input channel (0)	97	Ainput5	analog input channel (4)
48	Ainput2	analog input channel (1)	98	Ainput6	analog input channel (5)
49	Ainput3	analog input channel (2)	99	Ainput7	analog input channel (6)
50	Ainput4	analog input channel (3)	100	Ainput8	analog input channel (7)

*Mikroschritt möglich mit entsprechenden Mikroschritttreibern

¤

Analog Eingang Pinout Beschreibung

Analog Eingang Pinout Beschreibung (nur gültig für POSYS® 800-B)

J7 Pin-Out	Beschreibung	Erklärung
47	Analog Input 1	Analog input für channel (0)
48	Analog Input 2	Analog input für channel (1)
49	Analog Input 3	Analog input für channel (2)
50	Analog Input 4	Analog input für channel (3)

J1 Pin-Out	Beschreibung	Erklärung
97	Analog Input 5	Analog input für channel (4)
98	Analog Input 6	Analog input für channel (5)
99	Analog Input 7	Analog input für channel (6)
100	Analog Input 8	Analog input für channel (7)

Analog1-8 sind 8 frei-programmierbare analoge Eingangssignale. Falls angeschlossen stellen die Spannungen an den Eingängen zwar noch nichts dar und der Motion Prozessor reagiert nicht darauf, allerdings stellen sie eine gute Möglichkeit dar, dass des Anwender's Programm interaktiv hierrauf reagiert. Diese Signale werden mit dem Kommando ReadAnalog gelesen.

Die minimal erlaubte Eingangsspannung beträgt 0,0V und die maximale erlaubte Eingangsspannung beträgt 4.096V. Die

Auflösung beträgt 10 Bit. Um den numerischen Wert zu bestimmen kann folgende Formel verwendet werden:

ReadValue(Channel) = (AnalogVoltage * 65,536) / 4.096V

Umgekehrt wird mit folgender Formel die entsprechende Spannung berechnet:

AnalogVoltage = (ReadValue(Channel) * 4.096V) / 65,536

Damit die Analogspannungen korrekt ermittelt werden können, müssen die Analogsignale zusätzlich mit AnalogGND (Analog Ground) verbunden sein.

Falls andere als die standardmäßigen Eingangsspannungen gewünscht sind, können nach Absprache folgende Spannungen ersatzweise und optional angeboten werden: 1.8V, 2.5V, 3.0V und 3.3V.

¤

POSYS® 18xx

Versionen 18xx

Liste der verfügbaren Versionen

POSYS® 182x für bürstenbehaftete/bürstenlose (interne und externe Kommutierung) Servomotoren, Mikroschritt und Schrittmotoren.

Die POSYS® 182x kann für alle Motortypen verwendet werden (bürstenbehaftete Servomotoren, bürstenlose Servomotoren (interne und externe Kommutierung, Mikroschrittmotoren und Schrittmotoren)

Modell	# Achsen
POSYS® 1822	2
POSYS® 1824	4

POSYS® 185x für Schrittmotoren

Die POSYS® 185x kann auch für Mikroschrittmotoren verwendet werden wenn Mikroschritttreiber verwendet werden.

Modell	# Achsen
POSYS® 1852	2
POSYS® 1854	4

Alle Versionen sind auch für den erweiterten Temperaturbereich erhältlich (-40° - +85°C). Nicht auf Lager. Für Bestellungen fügen sie der Bestellnummer ein "-I" hinzu (z.B.: 1824-I).

¤

Einstellbare Komponenten

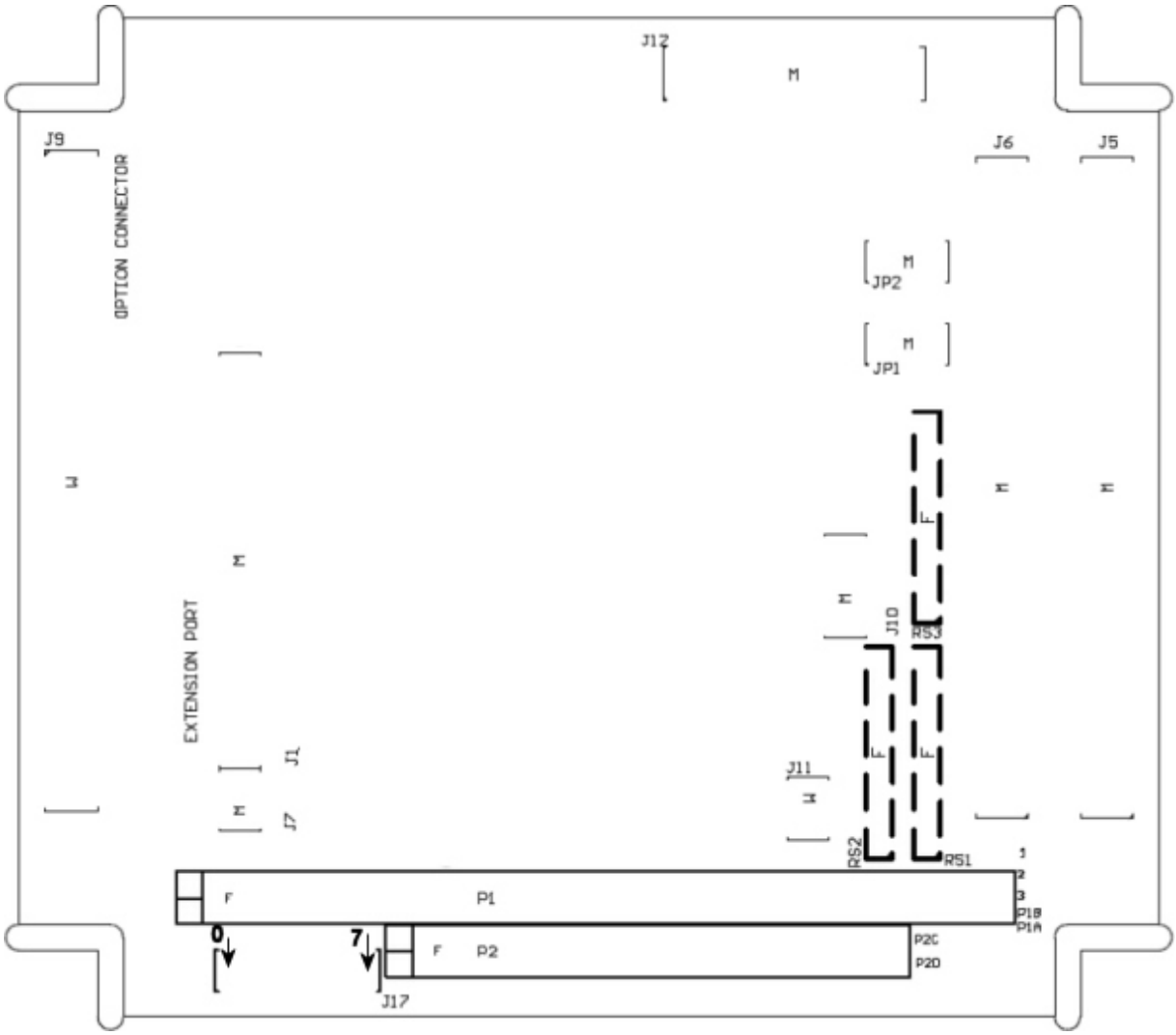
Einstellbare Komponenten; POSYS® 18xx

In diesem Abschnitt erhalten Sie die Informationen die notwendig sind alle Komponenten korrekt mit einander zu verbinden. Das folgende Bild zeigt die Position der einzelnen Verbindungen auf mit ihren Bezeichnungen zur leichteren Orientierung.

Die wichtigsten anwender bezogenen Komponenten sind:

- Widerstandsnetzwerke RS1, RS2 und RS3. Sie bestimmen die Art der Enkoderterminierung
- Dipschalter für Basisadresse und Basisspeicheradresse (SW1, SW2)
- Jumper für das Setzen des Interrupts (J17)
- JP1 und JP2 für das definieren des an der jeweiligen Achse befindlichen Motortyps (nicht 185x)
- Mode Jumper erlaubt die serielle Schnittstelle als primärer Kommunikationsport ausgewählt werden zu können. Ansonsten (Standard) ist es der PC-Bus. In diesem Modus ist die POSYS 18xx auch als Standalone Controller verwendbar
- CAN 2.0B erlaubt die CAN Schnittstelle als primären Kommunikationsport zu verwenden. In diesem Modus ist die POSYS 18xx auch als Standalone Controller verwendbar
- Sync I/O um die Karte in den Master oder Slave Modus zu schalten, falls weitere Karten im System eingesetzt werden.

© POSYS Motion Control GmbH & Co.KG, 2013



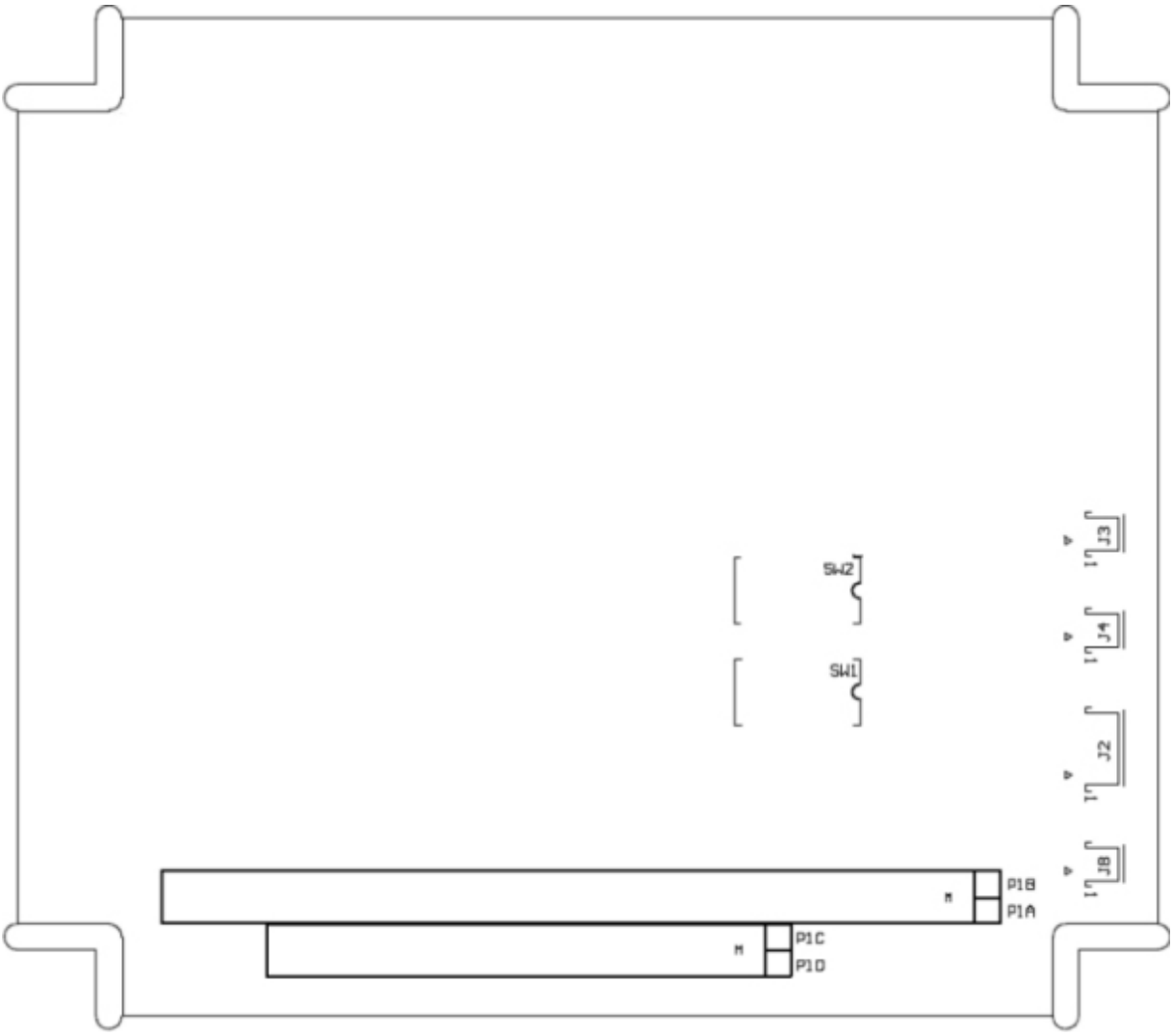


Abbildung zeigt POSYS 18xx

Widerstandsnetzwerke

Die folgende Tabelle zeigt die Funktionen der Widerstandsnetzwerke RS1-RS3 auf:

Komponente	Einstellung	Beschreibung
Netzwerk-widerstände	Installiert: RS1-RS3 (Standard)	Für die Verwendung von Enkodern mit invertierten Signalen sollten die Widerstandsnetzwerke installiert bleiben
	Deinstalliert: RS1-RS3	Für Enkoder mit TTL-Signalausgabe
	Sollten beide Versionen von Enkodern im System eingesetzt werden, ist es auch möglich die Widerstandsnetzwerke entsprechend anzupassen. Bitte setzen Sie sich mit uns in Verbindung für die genauen Pinpositionen der Widerstandsnetzwerke	

Falls sowohl Enkoder mit TTL als auch Enkoder mit invertierten Signalen Verwendung finden, können Sie auf der Seite Resistor Networks for TTL or inverted Encoder Signals Detailinformationen finden wie die Widerstandsnetzwerke einzustellen sind.

Dipschalter und Jumper Stellungen

Die POSYS 18xx habt Dipschalter für Adressinformationen und Steckbrücken um bestimmte Funktionalitäten zu de- oder aktivieren. Diese Stellungen sind in der folgenden Tabelle näher beschrieben:

Komponente	Einstellung	Beschreibung
SW1	S1-1 8 (hex) S1-2 10 (hex) S1-3 20 (hex) S1-4 40 (hex) S1-5 80 (hex) S1-6 100 (hex) S1-7 200 (hex) S1-8 400 (hex) 300 (hex) ist die Standardeinstellung	Switch S1 setzt die Kartenadresse für den ISA-Bus. Die gewählte Adresse ist der kumulierte Wert aus den entsprechenden Einstellungen wie sie links beschrieben sind. Ist der Schalter in der AUS-Position wird der Wert zum endgültigen Wert hinzu gerechnet. z.B um den Wert 340 (hex) einzustellen würden die Schalter auf folgenden Stellungen gesetzt: S1-1 on S1-2 on S1-3 on S1-4 off S1-5 on S1-6 off S1-7 off S1-8 on Die Kartenadresse eingestellt auf der Karte muss genau der Adresse in Ihrer Software entsprechen. Ausserdem darf diese Adresse von keiner anderen Karte belegt sein. Ausgehend von der Basisadresse belegt die POSYS anschliessend 8 Adressen. z.B. falls 300 (hex) eingestellt wurde, werden die Adressen von 300-307 (hex) belegt.
SW 2	S2-1 8 (hex) S2-2 10 (hex) S2-3 20 (hex) S2-4 40 (hex) S2-5 80 (hex) S2-6 100 (hex) S2-7 200 (hex) S2-8 400 (hex) D000 ist die Standardeinstellung	Wird nur benötigt wenn die Karte mit Dual Port RAM bestückt ist um den zu mappenden Memorybereich zu definieren. Standardmäßig wird der Memorybereich auf D000 gemappt.
Jumper 17	Nicht gesteckt (standard)	Mit den Steckbrücken #17 kann der gewünschte zu verwendende Interrupt selektiert werden. Nur ein Interrupt kann ausgewählt werden und dieser Interrupt darf auch sonst nicht von anderen Komponenten bereits belegt sein. Um z.B. IRQ3 auszuwählen muss die Steckbrücke 0 gesteckt sein. J17 Designation — ISA IRQ No. 0 — 3 1 — 5 2 — 7 3 — 10 4 — 11 5 — 12 6 — 14 7 — 15

Mode Jumper

Die folgende Tabelle zeigt die Funktionen des Mode Jumpers auf.

Komponente	Einstellung	Beschreibung
Mode Jumper	1-2 (Standard)	Der PC-Bus ist in dieser Stellung als Hauptkommunikationsleitung definiert. Die serielle Schnittstelle ist alternativ als Kommunikationsport ohne Einschränkung verwendbar.
	2-3	Der PC-Bus als Kommunikationsleitung ist deaktiviert, es ist nur noch möglich über die serielle Schnittstelle (RS232 = Standard; RS422/485 über optionale Module zusätzlich möglich) zu kommunizieren.

¤

SYNC IO

SYNC I/O

Diese beiden Stecker erlauben es mehrere im System befindliche Karten mit einander zu synchronisieren, sodass sie innerhalb des gleichen Arbeitszyklus arbeiten. Falls diese Verbindung nicht verwendet wird, fangen die anderen Karten erst mit ihrer Arbeit an nachdem die 1. Karte initialisiert wurde. Dies kann 256 µs nach einer 4-Achsen Karte sein. Mit aktiviertem SYNC I/O würden alle zusätzlichen Karten innerhalb von 50 ns nach dem Master anfangen zu arbeiten. Dies erlaubt präzises synchronisieren aller implementierten Achsen. Das Kommando Set/GetSynchronizationMode erklärt die Programmierung dieses Modus. Zusätzlich benötigen Sie das optionale SYNC I/O Kabel (1 für jede Kombination von 2 Karten).

Die Steckverbindung auf der Karte ist ein Molex Stecker (Artikelnummer: 53261-0271). Das dazu gehörige Gegenstück ist eine Molex Buchse (Artikelnummer: Molex 51021-0200, Crimpkontakt: Molex 50079-8100).

Set/GetSynchronizationMode, Sync IO Description

¤

Serielle Schnittstelle

Serial I/F Stecker PC/104-Karte (POSYS 18xx)

Nur gültig für die POSYS 18xx Serie

Dieser Stecker stellt eine serielle Verbindung zur POSYS 18xx zur Verfügung. Der Serial I/F Stecker ist ein 5-poliger Molex MLX Micro-4-SMD Stecker

Pin	Verbindung	Beschreibung
1	SrlEnable	Serial enable (wird nur für RS422/485 benötigt)
2	SrlXmt	Serial transmit output
3	SrlRcv	Serial receive input
4	GND	Ground
5	Vcc	+5V

Diese Schnittstelle erlaubt die POSYS 18xx über die serielle Schnittstelle RS232 zu betreiben. Sie kann als primäre oder sekundäre (z.B für Monitoringaufgaben) Kommunikationsschnittstelle verwendet werden.

Das serielle Kabel ist optional. Die folgende Beschreibung listet die notwendigen Komponenten auf falls es gewünscht ist ein eigenes serielles Kabel zu verwenden.
5-poliges Molex MicroBlade SMD Stecker auf der Karte (Molex Bestellnummer: 53398-0571)
Gegenstück für Kabel Molex Bestellnummer: 51021-0500

Die Standardeinstellungen für die serielle Kommunikation nach dem Einschalten sind:
Geschwindigkeit: 57200 Baud/sek.
Parität: keine
Stop Bit: 2
Protokoll: point-to-point

Falls andere Standardeinstellungen gewünscht werden, dann setzen Sie sich bitte mit POSYS Motion Control GmbH & Co.KG in Verbindung. Diese Werte sind aus kompatibilitätsgründen gewählt worden, können aber kundenspezifisch angepasst werden. Beim Einschalten würden diese neuen Werte sofort effektiv.

Set/GetSerialPortMode, Serial Port Description

¤

CAN 2.0B

CAN 2.0B

Diese Verbindung erlaubt die Kommunikation mit der POSYS® via CAN 2.0B. In diesem Modus kann die POSYS® 1900/1800 auch als Standalone Controller verwendet. Es werden nur 5V Spannungsversorgung benötigt. Um die POSYS® in CAN 2.0B Modus als Standalone Controller verwenden zu können, können wir spezielle Micro Boxes anbieten, welche, je nach Modell, PCI- bzw. ISA-Slots bieten um die Karten mit der erfordelichen Spannung zu versorgen. Zusätzliche CPU-Karten sind nicht notwendig.

Diese Methode erlaubt es bis zu 128 „Nodes“ (Knoten) zu verbinden und somit bis zu 512 Achsen zu kombinieren.

Feldbusse erlauben leider in dem Maße keine Hochgeschwindigkeitskommunikation für präzise Interpolation. Die Kombination mehrerer Achsen auf einer Karte welches via CAN 2.0B mit einem Host kommuniziert, erlaubt aber nun genau dies – präzise Interpolation (linear und zirkular (Option)) auf einem Feldbus.

Das CAN 2.0B Kabel ist optional. Bitte kontaktieren Sie uns falls Sie es wünschen die POSYS® im CAN Modus zu betreiben.

Die Steckverbindung auf der Karte ist ein Molex Stecker (Artikelnummer: 53261–0271). Das dazu gehörige Gegenstück ist eine Molex Buchse (Artikelnummer: Molex 51021-0200, Crimpkontakt: Molex 50079-8100).

Für weitere Information verweisen wir auch auf folgende Seiten: CAN (Controller Area Network, Konfiguration der CAN2.0B Schnittstelle, CAN Ereignis Benachrichtigung and Set/GetCanMode)

¤

Motor Mode Konfiguration; POSYS® 18xx

Motor Mode Jumpers „JP1“ und „JP2“

Je nach Modell gibt es verschiedene Möglichkeiten die Achsen der POSYS® für die Verwendung mit bestimmten Motortypen, wie z.B. bürstenbehaftete/bürstenlose (mit und ohne On-Board Kommutierung) Servomotoren, Mikroschritt oder Schrittmotoren zu verwenden. Neben dem Hardwaresetup für verschiedene Motortypen, sind auch verschiedene Softwareeinstellungen vor zu nehmen. Die Softwareeinstellungen sind im Programmierhandbuch beschrieben. Die Standardeinstellung für jede Achse nach dem Einschalten ist generell „bürstenbehafteter Servomotor“, ausser für die Schrittmotorversion der Karte. Dann ist generell „Schrittmotor“ eingestellt. Falls Sie eine andere Standardeinstellung nach dem Einschalten wünschen, dann teilen Sie uns das bitte mit (z.B.: 2 Achsen = DC Brushless, 1 Achse = Schritt, 1 Achse = DC Bürstenbehaftet). Diese Einstellungen können fix auf der Karte kodiert werden.

Die folgende Sektion beschreibt nur die Hardwarekonfiguration für den einen oder anderen Motortyp, eine gemischte Stellung ist möglich. Bedenken Sie aber, dass dann die jeweilige Achse für die eine oder andere Methode konfiguriert ist. Um dies wieder zu ändern, müssen die Steckbrücken JP1 und JP2 erneut ihren Erfordernissen angepasst werden:

Motortyp	Jumper Stellung
DC Brushed	JP2
DC Brushless	JP2
Mikroschritt	JP2
Schritt	JP1

JP1 und JP2 bestehen beide aus 4 Steckbrücken. Jede Steckbrücke steht für Achse 1, 2, 3 oder 4 und ist auch abhängig davon ob Sie eine 1, 2, 3 oder 4 Achsen Karte erworben haben. Auch hängen die Möglichkeiten von der Kartenversion ab. Die folgende Tabelle beschreibt welche Steckbrückenposition notwendig ist um das korrekte Motorausgangssignal für den angeschlossenen Verstärker/Motor zu erhalten.

POSYS® 182x

Die POSYS® 182x kann für die Verwendung mit DC bürstenbehafteten Servomotoren, bürstenlosen Servomotoren mit interner/externer Kommutierung, Mikroschritt oder Schrittmotor eingestellt werden. Die folgende Tabelle beschreibt die korrekte Positionierung der Steckbrücken JP1 und JP2 für die eine oder andere Motorversion:

Achse	Motortyp	
	DC Bürstenbehafteter Servomotor DC Bürstenloser Servomotor (mit interner/externer Kommutierung) oder Mikroschrittmotor	Schritt- oder Mikroschrittmotor (mit Mikroschritttreiber)

1	JP2 kurz (erste von links); JP1 offen	JP1 kurz (erste von links); JP2 offen
2	JP2 kurz (zweite von links); JP1 offen	JP1 kurz (zweite von links); JP2 offen
3	JP2 kurz (dritte von links); JP1 offen	JP1 kurz (dritte von links); JP2 offen
4	JP2 kurz (vierte von links); JP1 offen	JP1 kurz (vierte von links); JP2 offen

POSYS® 185x

Die POSYS® 185x kann nur für die Verwendung mit Schrittmotoren eingerichtet werden. Die Verwendung von Servomotorverstärker ist aber trotzdem dann möglich. Es gibt einige Hersteller (Yaskawa, Mitsubishi, Panasonic, Tamagawa usw.) die Verstärker mit interner PI- oder PID-Regelung herstellen und das Takt- & Richtungssignal der POSYS® in entsprechende Bewegung umsetzen. Bei Verwendung des entsprechenden Treibers ist auch der Einsatz von Mikroschrittmotoren möglich. Die folgende Tabelle beschreibt die korrekte Positionierung der Steckbrücken JP1 und JP2 für die eine oder andere Motorversion:

Achse	Motortyp	
	DC Bürstenbehafteter Servomotor DC Bürstenloser Servomotor (mit interner/externer Kommutierung) oder Mikroschrittmotor	Schritt- oder Mikroschrittmotor (mit Mikroschritttreiber)
1	n/a	JP1 kurz (erste von links); JP2 offen
2	n/a	JP1 kurz (zweite von links); JP2 offen
3	n/a	JP1 kurz (dritte von links); JP2 offen
4	n/a	JP1 kurz (vierte von links); JP2 offen

¤

Bürstenbehaftete Servomotor Pinout Beschreibung

Pinout Beschreibung für bürstenbehaftete oder bürstenlose (mit externe Kommutierung) Servomotoren

POSYS® 182x
Pin 1-50 = J5, Pin 51-100 = J6

Verbindungen auf der Karte:
TSM-125-01-T-DV (Samtec)
Gegenstecker: IDSD-25-S

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
1	A1+	Channel A+ encoder signal (X)	51	A3+	Channel A+ encoder signal (Z)
2	A1-	Channel A- inverted encoder signal (X)	52	A3-	Channel A- inverted encoder signal (Z)
3	B1+	Channel B+ encoder signal (X)	53	B3+	Channel B+ encoder signal (Z)
4	B1-	Channel B- inverted encoder signal (X)	54	B3-	Channel B- inverted encoder signal (Z)
5	I1+	Channel I+ encoder signal (X)	55	I3+	Channel I+ encoder signal (Z)
6	I1-	Channel I- inverted encoder signal (X)	56	I3-	Channel I- inverted encoder signal (Z)
7	Vc1	+5V	57	Vc1	+5V
8	GND	Ground	58	GND	Ground
9	XPLIM1	Limit switch positive direction (X)	59	XPLIM3	Limit switch positive direction (Z)
10	XNLIM1	Limit switch negative direction (X)	60	XNLIM3	Limit switch negative direction (Z)
11	HOME1	Home signal (X) or high speed latch	61	HOME3	Home signal (Z) or high speed latch
12	GND	Ground	62	GND	Ground

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
13	OUT1	User programmable output (X)	63	OUT3	User programmable output (Z)
14	BPWM1	PWM magnitude (X)	64	BPWM3	PWM magnitude (Z)
15	BPWMS1	PWM sign (X)	65	BPWMS3	PWM sign (Z)
16	AXIN1	User programmable input (X)	66	AXIN3	User programmable input (Z)
17	DACV1	Motor command (X), ±10V signal	67	DACV3	Motor command (Z), ±10V signal
18	AGND	Ground for motor command	68	AGND	Ground for motor command
19	A2+	Channel A+ encoder signal (Y)	69	A4+	Channel A+ encoder signal (W)
20	A2-	Channel A- inverted encoder signal (Y)	70	A4-	Channel A- inverted encoder signal (W)
21	B2+	Channel B+ encoder signal (Y)	71	B4+	Channel B+ encoder signal (W)
22	B2-	Channel B- inverted encoder signal (Y)	72	B4-	Channel B- inverted encoder signal (W)
23	I2+	Channel I+ encoder signal (Y)	73	I4+	Channel I+ encoder signal (W)
24	I2-	Channel I- inverted encoder signal (Y)	74	I4-	Channel I- inverted encoder signal (W)
25	Vc1	+5V	75	Vc1	+5V
26	GND	Ground	76	GND	Ground
27	XPLIM2	Limit switch positive direction (Y)	77	XPLIM4	Limit switch positive direction (W)
28	XNLIM2	Limit switch negative direction (Y)	78	XNLIM4	Limit switch negative direction (W)
29	HOME2	Home signal (Y) or high speed latch	79	HOME4	Home signal (W) or high speed latch
30	OUT2	User programmable output (Y)	80	OUT4	User programmable output (W)
31	BPWM2	PWM magnitude (Y)	81	BPWM4	PWM magnitude (W)
32	BPWMS2	PWM sign (Y)	82	BPWMS4	PWM sign (W)
33	AXIN2	User programmable input (Y)	83	AXIN4	User programmable input (W)
34	DACV2	Motor command (Y), ±10V signal	84	DACV4	Motor command (W), ±10V signal
35	AGND	Ground for motor command	85	AGND	Ground for motor command
36	UI0	uncommitted input (0)	86	UI4	uncommitted input (4)
37	UI1	uncommitted input (1)	87	UI5	uncommitted input (5)
38	UI2	uncommitted input (2)	88	UI6	uncommitted input (6)
39	UI3	uncommitted input (3)	89	UI7	uncommitted input (7)
40	AMPEN1	Amplifier enable (X) or user programmable output	90	AMPEN3	Amplifier enable (Z) or user programmable output
41	UO0	uncommitted output (0)	91	UO4	uncommitted output (4)
42	UO1	uncommitted output (1)	92	UO5	uncommitted output (5)
43	UO2	uncommitted output (2)	93	UO6	uncommitted output (6)
44	UO3	uncommitted output (3)	94	UO7	uncommitted output (7)
45	AMPEN2	Amplifier enable (Y) or user programmable output	95	AMPEN4	Amplifier enable (W) or user programmable output
46	RS-	Hardware reset	96	ANGND	GND for analog inputs
47	Ainput1	analog input channel (0)	97	Ainput5	analog input channel (4)
48	Ainput2	analog input channel (1)	98	Ainput6	analog input channel (5)
49	Ainput3	analog input channel (2)	99	Ainput7	analog input channel (6)
50	Ainput4	analog input channel (3)	100	Ainput8	analog input channel (7)

Signalbelegung J12 falls mit Schritt- (1. Option) oder Servomotor (2. Option) betrieben (je nach Motor Mode Konfiguration für jede Achse indiv iduell)

TSM-110-01-T-DV (Samtec)

Dieser Stecker steht auf allen POSYS 1800 zur Verfügung. Es stellt invertierte Signalausgänge für Schrittmotoren (Takt- & Richtung) und invertierte 1-phasige PWM Sign/Mag Ausgabesignale über Line Driver zur Verfügung. Jede Achse ist individuell einstellbar, je nachdem wleche Motorausgabemethode und Motortype gewählt wurde. Die meisten modernen Schrittmotorentreiber erfordern invertierte Takt- & Richtungssignale. Es unterstützt die Signalintegrität auch in störempfindlichen Umgebungen. Die Gefahr des Verlustes an Takt- & Richtungssignalen (gleichbedeutend mit Positionierungenauigkeit) wird erheblich verringert.

Abhängig von der gewählten Motortype der Achse und der gewählten Motorsignalausgabemethode, haben die einzelnen Anschlüsse unterschiedliche Bedeutungen.

Pin	Beschreibung	Pin	Beschreibung
1	GND	2	GND
3	W axis, Pulse + or	4	W axis, Pulse - or
	PWMMag4A+		PWMMag4A-
5	X axis, Direction + or	6	X axis, Direction - or
	PWMSign1+		PWMSign1-
7	W axis, Direction + or	8	W axis, Direction - or
	PWMSign4+		PWMSign4-
9	Y axis, Pulse + or	10	Y axis, Pulse - or
	PWMMag2A+		PWMMag2A-
11	Z axis, Direction + or	12	Z axis, Direction - or
	PWMSign3+		PWMSign3-
13	X axis, Pulse + or	14	X axis, Pulse - or
	PWMMag1A+		PWMMag1A+
15	Y axis, Direction + or	16	Y axis, Direction - or
	PWMSign2+		PWMSign2-
17	Z axis, Pulse + or	18	Z axis, Pulse - or
	PWMMag3A+		PWMMagA3-
19	GND	20	GND

¤

Bürstenlose Servo- und Mikroschrittmotor Pinout Beschreibung

Pinout Beschreibung bürstenlose Servomotoren mit interne Kommutierung, Mikroschritt und Multiple Motor Modus

POSYS® 182x
Pin 1-50 = J5, Pin 51-100 = J6

Verbindungen auf der Karte:
TSM-125-01-T-DV (Samtec)
Gegenstecker: IDSD-25-S

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
1	A1+	Channel A+ encoder signal (X)	51	A3+	Channel A+ encoder signal (Z)
2	A1-	Channel A- inverted encoder signal (X)	52	A3-	Channel A- inverted encoder signal (Z)
3	B1+	Channel B+ encoder signal (X)	53	B3+	Channel B+ encoder signal (Z)
4	B1-	Channel B- inverted encoder signal (X)	54	B3-	Channel B- inverted encoder signal (Z)
5	I1+	Channel I+ encoder signal (X)	55	I3+	Channel I+ encoder signal (Z)
6	I1-	Channel I- inverted encoder	56	I3-	Channel I- inverted encoder

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
		signal (X)			signal (Z)
7	Vc1	+5V	57	Vc1	+5V
8	GND	Ground	58	GND	Ground
9	XPLIM1	Limit switch positive direction (X)	59	XPLIM3	Limit switch positive direction (Z)
10	XNLIM1	Limit switch negative direction (X)	60	XNLIM3	Limit switch negative direction (Z)
11	HOME1	Home signal (X) or high speed latch	61	HOME3	Home signal (Z) or high speed latch
12	GND	Ground	62	GND	Ground
13	OUT1	User programmable output (X)	63	OUT3	User programmable output (Z)
14	n.c.	No connection.	64	n.c.	No connection
15	n.c.	No connection	65	n.c.	No connection
16	AXIN1	User programmable input (X)	66	AXIN3	User programmable input (Z)
17	DACV1	Motor command (X), ±10V signal	67	DACV3	Motor command (Z), ±10V signal
18	AGND	Ground for motor command	68	AGND	Ground for motor command
19	A2+	Channel A+ encoder signal (Y)	69	A4+	Channel A+ encoder signal (W)
20	A2-	Channel A- inverted encoder signal (Y)	70	A4-	Channel A- inverted encoder signal (W)
21	B2+	Channel B+ encoder signal (Y)	71	B4+	Channel B+ encoder signal (W)
22	B2-	Channel B- inverted encoder signal (Y)	72	B4-	Channel B- inverted encoder signal (W)
23	I2+	Channel I+ encoder signal (Y)	73	I4+	Channel I+ encoder signal (W)
24	I2-	Channel I- inverted encoder signal (Y)	74	I4-	Channel I- inverted encoder signal (W)
25	Vc1	+5V	75	Vc1	+5V
26	GND	Ground	76	GND	Ground
27	XPLIM2	Limit switch positive direction (Y)	77	XPLIM4	Limit switch positive direction (W)
28	XNLIM2	Limit switch negative direction (Y)	78	XNLIM4	Limit switch negative direction (W)
29	HOME2	Home signal (Y) or high speed latch	79	HOME4	Home signal (W) or high speed latch
30	OUT2	User programmable output (Y)	80	OUT4	User programmable output (W)
31	n.c.	No connection	81	n.c.	No connection
32	n.c.	No connection	82	n.c.	No connection
33	AXIN2	User programmable input (Y)	83	AXIN4	User programmable input (W)
34	DACV2	Motor command (Y), ±10V signal	84	DACV4	Motor command (W), ±10V signal
35	AGND	Ground for motor command	85	AGND	Ground for motor command
36	UI0	uncommitted input (0)	86	UI4	uncommitted input (4)
37	UI1	uncommitted input (1)	87	UI5	uncommitted input (5)
38	UI2	uncommitted input (2)	88	UI6	uncommitted input (6)
39	UI3	uncommitted input (3)	89	UI7	uncommitted input (7)
40	AMPEN1	Amplifier enable (X) or user programmable output	90	AMPEN3	Amplifier enable (Z) or user programmable output
41	UO0	uncommitted output (0)	91	UO4	uncommitted output (4)
42	UO1	uncommitted output (1)	92	UO5	uncommitted output (5)
43	UO2	uncommitted output (2)	93	UO6	uncommitted output (6)
44	UO3	uncommitted output (3)	94	UO7	uncommitted output (7)
45	AMPEN2	Amplifier enable (Y) or user programmable output	95	AMPEN4	Amplifier enable (W) or user programmable output

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
46	RS-	Hardware reset	96	ANGND	GND for analog inputs
47	Ainput1	analog input channel (0)	97	Ainput5	analog input channel (4)
48	Ainput2	analog input channel (1)	98	Ainput6	analog input channel (5)
49	Ainput3	analog input channel (2)	99	Ainput7	analog input channel (6)
50	Ainput4	analog input channel (3)	100	Ainput8	analog input channel (7)

Signalbelegung J12 falls mit Schritt- (1. Option) oder Servomotor (2. Option) betrieben (je nach Motor Mode Konfiguration für jede Achse indiv iduell)

TSM-110-01-T-DV (Samtec)

Dieser Stecker steht auf allen POSYS 1800 zur Verfügung. Es stellt invertierte Signalausgänge für Schrittmotoren (Takt- & Richtung) und invertierte 1-phasige PWM Sign/Mag Ausgabesignale über Line Driver zur Verfügung. Jede Achse ist individuell einstellbar, je nachdem wleche Motorausgabemethode und Motortype gewählt wurde. Die meisten modernen Schrittmotorentreiber erfordern invertierte Takt- & Richtungssignale. Es unterstützt die Signalintegrität auch in störempfindlichen Umgebungen. Die Gefahr des Verlustes an Takt- & Richtungssignalen (gleichbedeutend mit Positionierungenauigkeit) wird erheblich verringert.

Abhängig von der gewählten Motortype der Achse und der gewählten Motorsignalausgabemethode, haben die einzelnen Anschlüsse unterschiedliche Bedeutungen.

Pin	Beschreibung	Pin	Beschreibung
1	GND	2	GND
3	W axis, Pulse + or PWMMag4A+	4	W axis, Pulse - or PWMMag4A-
5	X axis, Direction + or PWMSign1+	6	X axis, Direction - or PWMSign1-
7	W axis, Direction + or PWMSign4+	8	W axis, Direction - or PWMSign4-
9	Y axis, Pulse + or PWMMag2A+	10	Y axis, Pulse - or PWMMag2A-
11	Z axis, Direction + or PWMSign3+	12	Z axis, Direction - or PWMSign3-
13	X axis, Pulse + or PWMMag1A+	14	X axis, Pulse - or PWMMag1A+
15	Y axis, Direction + or PWMSign2+	16	Y axis, Direction - or PWMSign2-
17	Z axis, Pulse + or PWMMag3A+	18	Z axis, Pulse - or PWMMagA3-
19	GND	20	GND

Pinout Beschreibung bürstenlose Servomotoren mit interne Kommutierung, Mikroschritt und Multiple Motor Modus; Option Connector PC/104

Anmerkung: nur gültig für POSYS® 182x (J9)

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
1	BPWM1A	PWM magnitude Phase A (X)	26	HALL2B	Phase B Hall Sensor input (Y)
2	BPWM1B	PWM magnitude Phase B (X)	27	HALL2C	Phase C Hall Sensor input (Y)
3	BPWM1C	PWM magnitude Phase C (X)	28	GND	Ground
4	BPWMS1A	PWM sign (X)	29	HALL3A	Phase A Hall Sensor input (Z)
5	GND	Ground	30	HALL3B	Phase B Hall Sensor input (Z)

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
6	BPWM2A	PWM magnitude Phase A (Y)	31	HALL3C	Phase C Hall Sensor input (Z)
7	BPWM2B	PWM magnitude Phase B (Y)	32	GND	Ground
8	BPWM2C	PWM magnitude Phase C (Y)	33	HALL4A	Phase A Hall Sensor input (W)
9	BPWMS2A	PWM sign (Y)	34	HALL4B	Phase B Hall Sensor input (W)
10	GND	Ground	35	HALL4C	Phase C Hall Sensor input (W)
11	BPWM3A	PWM magnitude Phase A (Z)	36	GND	Ground
12	BPWM3B	PWM magnitude Phase B (Z)	37	AGND	Ground for motor command
13	BPWM3C	PWM magnitude Phase C (Z)	38	DAC1A	Motor command (X), ±10V signal, Phase A
14	BPWMS3A	PWM sign (Z)	39	DAC2A	Motor command (Y), ±10V signal, Phase A
15	GND	Ground	40	DAC3A	Motor command (Z), ±10V signal, Phase A
16	BPWM4A	PWM magnitude Phase A (W)	41	DAC4A	Motor command (W), ±10V signal, Phase A
17	BPWM4B	PWM magnitude Phase B (W)	42	DAC1B	Motor command (X), ±10V signal, Phase B
18	BPWM4C	PWM magnitude Phase C (W)	43	DAC2B	Motor command (Y), ±10V signal, Phase B
19	BPWMS4A	PWM sign (W)	44	DAC3B	Motor command (Z), ±10V signal, Phase B
20	GND	Ground	45	DAC4B	Motor command (W), ±10V signal, Phase B
21	HALL1A	Phase A Hall Sensor input (X)	46	AGND	Ground for motor command
22	HALL1B	Phase B Hall Sensor input (X)	47	GND	Ground
23	HALL1C	Phase C Hall Sensor input (X)	48	GND	Ground
24	GND	Ground	49	Vc1	+5V
25	HALL2A	Phase A Hall Sensor input (Y)	50	Vc1	+5V

¤

Schrittmotor Pinout Beschreibung

Pinout Beschreibung Schritt-/Mikroschrittmotor* Version

POSYS® 185x
Pin 1-50 = J5, Pin 51-100 = J6

Verbindungen auf der Karte:
TSM-125-01-T-DV (Samtec)
Gegenstecker: IDSD-25-S

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
1	A1+	Channel A+ encoder signal (X)	51	A3+	Channel A+ encoder signal (Z)
2	A1-	Channel A- inverted encoder signal (X)	52	A3-	Channel A- inverted encoder signal (Z)
3	B1+	Channel B+ encoder signal (X)	53	B3+	Channel B+ encoder signal (Z)
4	B1-	Channel B- inverted encoder signal (X)	54	B3-	Channel B- inverted encoder signal (Z)
5	I1+	Channel I+ encoder signal (X)	55	I3+	Channel I+ encoder signal (Z)
6	I1-	Channel I- inverted encoder signal (X)	56	I3-	Channel I- inverted encoder signal (Z)
7	Vc1	+5V	57	Vc1	+5V
8	GND	Ground	58	GND	Ground
9	XPLIM1	Limit switch positive direction (X)	59	XPLIM3	Limit switch positive direction (Z)

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
10	XNLIM1	Limit switch negative direction (X)	60	XNLIM3	Limit switch negative direction (Z)
11	HOME1	Home signal (X) or high speed latch	61	HOME3	Home signal (Z) or high speed latch
12	GND	Ground	62	GND	Ground
13	OUT1	User programmable output (X)	63	OUT3	User programmable output (Z)
14	PULSE1	Pulse (X)	64	PULSE3	Pulse (Z)
15	DIR1	Direction (X)	65	DIR3	Direction (Z)
16	AXIN1	User programmable input (X)	66	AXIN3	User programmable input (Z)
17	ATREST1	At Rest (X)	67	ATREST3	At Rest (Z)
18	GND	Ground	68	GND	Ground
19	A2+	Channel A+ encoder signal (Y)	69	A4+	Channel A+ encoder signal (W)
20	A2-	Channel A- inverted encoder signal (Y)	70	A4-	Channel A- inverted encoder signal (W)
21	B2+	Channel B+ encoder signal (Y)	71	B4+	Channel B+ encoder signal (W)
22	B2-	Channel B- inverted encoder signal (Y)	72	B4-	Channel B- inverted encoder signal (W)
23	I2+	Channel I+ encoder signal (Y)	73	I4+	Channel I+ encoder signal (W)
24	I2-	Channel I- inverted encoder signal (Y)	74	I4-	Channel I- inverted encoder signal (W)
25	Vc1	+5V	75	Vc1	+5V
26	GND	Ground	76	GND	Ground
27	XPLIM2	Limit switch positive direction (Y)	77	XPLIM4	Limit switch positive direction (W)
28	XNLIM2	Limit switch negative direction (Y)	78	XNLIM4	Limit switch negative direction (W)
29	HOME2	Home signal (Y) or high speed latch	79	HOME4	Home signal (W) or high speed latch
30	OUT2	User programmable output (Y)	80	OUT4	User programmable output (W)
31	PULSE2	Pulse (Y)	81	PULSE4	Pulse (W)
32	DIR2	Direction (Y)	82	DIR4	Direction (W)
33	AXIN2	User programmable input (Y)	83	AXIN4	User programmable input (W)
34	ATREST2	At Rest (Y)	84	ATREST4	At Rest (W)
35	GND	Ground	85	GND	Ground
36	UI0	uncommitted input (0)	86	UI4	uncommitted input (4)
37	UI1	uncommitted input (1)	87	UI5	uncommitted input (5)
38	UI2	uncommitted input (2)	88	UI6	uncommitted input (6)
39	UI3	uncommitted input (3)	89	UI7	uncommitted input (7)
40	ATREST1	At Rest (X)	90	ATREST3	At Rest (Z)
41	UO0	uncommitted output (0)	91	UO4	uncommitted output (4)
42	UO1	uncommitted output (1)	92	UO5	uncommitted output (5)
43	UO2	uncommitted output (2)	93	UO6	uncommitted output (6)
44	UO3	uncommitted output (3)	94	UO7	uncommitted output (7)
45	ATREST2	At Rest (Y)	95	ATREST4	At Rest (W)
46	RS-	Hardware reset	96	ANGND	GND for analog inputs
47	Ainput1	analog input channel (0)	97	Ainput5	analog input channel (4)
48	Ainput2	analog input channel (1)	98	Ainput6	analog input channel (5)
49	Ainput3	analog input channel (2)	99	Ainput7	analog input channel (6)
50	Ainput4	analog input channel (3)	100	Ainput8	analog input channel (7)

*Mikroschritt möglich mit entsprechenden Mikroschritttreibern

Signalbelegung J12

TSM-110-01-T-DV (Samtec)

Dieser Stecker ist auf allen POSYS 1800 Versionen verfügbar. Es stellt Takt- & Richtungssignale in Verbindung Line Drivers zur Verfügung. Die meisten modernen Schrittmotortreiber benötigen invertierte Takt- & Richtungssignale. Es unterstützt die Signalqualität zu verbessern und störende Signalspitzen, die ansonsten falsch interpretiert werden können, richtig zu interpretieren.

Pin	Beschreibung	Pin	Beschreibung
1	GND	2	GND
3	W axis, Pulse +	4	W axis, Pulse -
5	X axis, Direction +	6	X axis, Direction -
7	W axis, Direction +	8	W axis, Direction -
9	Y axis, Pulse +	10	Y axis, Pulse -
11	Z axis, Direction +	12	Z axis, Direction -
13	X axis, Pulse +	14	X axis, Pulse -
15	Y axis, Direction +	16	Y axis, Direction -
17	Z axis, Pulse +	18	Z axis, Pulse -
19	GND	20	GND

¤

Analog Eingang Pinout Beschreibung

Analog Eingang Pinout Beschreibung (nur gültig für POSYS® 1800)

J5 Pin-Out	Beschreibung	Erklärung
47	Analog Input 1	Analog input für channel (0)
48	Analog Input 2	Analog input für channel (1)
49	Analog Input 3	Analog input für channel (2)
50	Analog Input 4	Analog input für channel (3)

J6 Pin-Out	Beschreibung	Erklärung
47	Analog Input 5	Analog input für channel (4)
48	Analog Input 6	Analog input für channel (5)
49	Analog Input 7	Analog input für channel (6)
50	Analog Input 8	Analog input für channel (7)

Analog1-8 sind 8 frei-programmierbare analoge Eingangssignale. Falls angeschlossen stellen die Spannungen an den Eingängen zwar noch nichts dar und der Motion Prozessor reagiert nicht darauf, allerdings stellen sie eine gute Möglichkeit dar, dass des Anwender's Programm interaktiv hierrauf reagiert. Diese Signale werden mit dem Kommando ReadAnalog (Channel) gelesen.

Die minimal erlaubte Eingangsspannung beträgt 0,0V und die maximale erlaubte Eingangsspannung beträgt 2.048V. Die Auflösung beträgt 10 Bit. Um den numerischen Wert zu bestimmen kann folgende Formel verwendet werden:

ReadValue(Channel) = (AnalogVoltage * 65,536) / 2.048V

Umgekehrt wird mit folgender Formel die entsprechende Spannung berechnet:

AnalogVoltage = (ReadValue(Channel) * 2.048V) / 65,536

Damit die Analogspannungen korrekt ermittelt werden können, müssen die Analogsignale zusätzlich mit AnalogGND (Analog Ground) verbunden sein.

Falls andere als die standardmäßigen Eingangsspannungen gewünscht sind, können nach Absprache folgende Spannungen ersatzweise und optional angeboten werden: 1.8V, 2.5V, 3.0V und 3.3V.

¤

Verwendung des On-Board Dual Port RAMs mit der POSYS® 1800

POSYS® 1800 Serie (PC-104 Bus), Dual Port RAM

Dieser Abschnitt beschreibt die Verwendung der POSYS® 1800 mit Dual Port RAM. Die POSYS® 1800 kann optional mit Single Port RAM (512k x 8 und 1024k x 8) oder mit Dual Port RAM (16k x 16) bestückt werden. Eine Datenposition benötigt 4 Bytes (32 Bit). Daher ist der Dual Port RAM in der Lage bis zu 8191 DWORDS zu speichern. Applikationen die den schnellen Datenaustausch und -zugriff voraussetzen, sollten Einheiten mit Dual Port RAM verwenden. Das Lesen und Schreiben von Dual Port RAM geschieht wesentlich schneller und benötigt dafür ca. 2 Mikrosekunden für ein DWORD, wohingegen mit Single Port RAM das Schreiben und Lesen eine DWORDS ca. 52 Mikrosekunden benötigt. Dual Port RAM ist ca. 26x schneller als Single Port RAM.

Die Applikationen können Tracedaten durch den Motion Prozessor in den Dual Port RAM speichern, die wiederum kontinuierlich vom Hostprogramm über einen anderen Port gelesen werden. Die Speicherkapazität ist dadurch unbegrenzt. Eine weitere Applikation ist die Verwendung des Dual Port RAM als Datenspeicher für Bahnprofile. Dabei werden Informationen über Geschwindigkeit, Beschleunigung, Jerk, Position und ein Zeitmultiplikator für bis zu 4 Achsen im EXTERNAL Profile Mode im Speicher hinterlegt. Sehr komplexe Bewegungsabläufe können dadurch ermöglicht werden mit unbegrenzter Länge. Wenn der Buffer zu einem Ende kommt können die neuen Daten wieder am Anfang in den Buffer geschrieben werden.

Im Gegensatz zur PCI-Karte mit Dual Port RAM wird bei PC-104 (das Gleiche wie ISA-Bus) der Dual Port RAM in den UMB (Upper Memory Block) des ISA-Bus gemappt. Dies ist typischerweise der Bereich zwischen 640k und 1024k. Dies ist der einzige verfügbarer Bereich um Daten zu speichern und zu empfangen und vor Programmen und Datenspeicherung unterhalb 640k zu schützen. Der Bereich oberhalb der 640k ist in 6 Segmente je 64k aufgeteilt. Die Segemente sind wie folgt definiert:

A000 : 0000 ;der erste Teil 0xA000 ist der Segment (=40960 dezimal), der zweite Teil ist der Offset, im Bereich vom 0x0000 bis 0xFFFF (0 bis 65535 dezimal). Dies entspricht eine linearen Adresse 0xA000 x 0x010 = 0xA0000 plus Offset.

B000 : 0000
C000 : 0000
D000 : 0000
E000 : 0000
F000 : 0000

Nicht alle dieser Adressen stehen aber einer Add-On ISA-Bus (PC-104) Karte zur Verfügung.

Die nachfolgende Tabelle zeigt den Bereich des UMB und wie es von verschiedenen PC-Komponeten typischerweise verwendet wird:

Address	First 16K (x0000h-x3FFFh)	Second 16K (x4000h-x7FFFh)	Third 16K (x8000h-xBFFFh)	Fourth 16K (xC000h-xFFFFh)
A0000-AFFFFh	VGA Graphics Mode Video RAM			
B0000-BFFFFh	VGA Monochrome Text Mode Video RAM		VGA Color Text Mode Video RAM	
C0000-CFFFFh	VGA Video BIOS ROM		IDE Hard Disk BIOS ROM	Optional Adapter ROM BIOS or RAM UMBs
D0000-DFFFFh	Optional Adapter ROM BIOS or RAM UMBs			
E0000h-EFFFFh	System BIOS Plug and Play Extended Information			
F0000-FFFFFh	System BIOS ROM			

Dies ist nur ein Beispiel. Die tatsächliche Verwendung kann von System zu System variieren. Erfahrungsgemäß aber sind die nachfolgend aufgeführten Bereiche in der Regel frei:

B0000 – BFFFFh	
C0000 – CFFFFh	most common
D0000 – DFFFFh	most common
E0000 – EFFFFh	

Die obengenannten Bereiche sind 65535 Bytes groß, der Dual Port RAM der POSYS® 1800 benötigt davon die Hälfte (0x7FFF (32767) bytes).

Um die Verfügbarkeit zu bestimmen, wird ein Diagnostikprogramm benötigt um das System zu analysieren. Ein Programm das gute Dienste diesbezüglich leisten kann, aber nicht Teil der Distribution für die POSYS® 1800 ist, ist das Programm MSD.EXE von Microsoft. Das Programm ist Copyright geschützt und Teil der MSDOS 6.22 Distribution. Es ist auf der 2. Diskette von insgesamt 3 Disketten. Es liegt normalerweise auch den Windows 9x Distributionen bei, wird aber bei der Installation nicht automatisch auf dem Rechner aufgespielt. Es kann in den Verzeichnissen /TOOLS/OLDMSDOS=MSD.EXE v2.14 und /OTHER/MSD=MSD.EXE v2.13 gefunden werden. Es ist ein DOS-basiertes Programm und stellt nützliche Informationen über das System zusammen und das Untermenü MEMORY gibt Aufschluss darüber welcher der Speicherbereiche zwischen 640k und 1024k zur Verfügung stehen. MSD.EXE funktioniert auf DOS (MSDOS, DRDOS, PCDOS) Systemen und Windows 9x/ME. Auf Systemen mit Windows 2000/XP funktioniert es nicht.

Die passende Base Memory Adresse muss mit Dipschaltern (SW2) eingestellt werden. Der minimale Bereich der eingestellt werden kann ist 32767 Bytes. Die Schalter 1, 2 und 3 müssen dabei immer auf ON gestellt sein. Die Schalterstellungen von 4, 5, 6, 7 und 8 bestimmen die Base Memory Adresse. Die Schalterstellungen sind in der folgenden Tabelle beschrieben.

SW2 -S4	SW2-S5	SW2-S6	SW2-S7	SW2-S8
32767 (0x8000)	65536 (0x10000)	131072 (0x20000)	262144 (0x40000)	524288 (0x80000)

Die folgende Tabelle zeigt Schalterstellungen für die geläufigsten Base Address Einstellungen:

Base Memory Range (Hex)	SW2-S4	SW2-S5	SW2-S6	SW2-S7	SW2-S8
B0000 – B7FFF	ON	OFF	OFF	ON	OFF
B8000 - BFFFF	OFF	OFF	OFF	ON	OFF
C0000 – C7FFF	ON	ON	ON	OFF	OFF
C8000 - CFFFF	OFF	ON	ON	OFF	OFF
D0000 – D7FFF	ON	OFF	ON	OFF	OFF
D8000 – DFFFF	OFF	OFF	ON	OFF	OFF
E0000 – E7FFF	ON	ON	OFF	OFF	OFF
E8000 – EFFFF	OFF	ON	OFF	OFF	OFF

Für DOS-basierte Programme verweisen wir auf das Beispielprogramm Test1800.exe, das zeigt wie Dual Port RAM gelesen und beschrieben werden kann. Es ist ein C-Programm und verwendet die Funktionen Peek und Poke. z.B. die folgende Prozedur schreibt 0 bis finalpos DWORDs:

```
ofs=0;
for (ii=0; ii<(finalpos+1); ii++)
{
   poke(base_memory,ofs,((WORD2LONG*)&value)->wordhi);
   ofs=ofs+2;
   poke(base_memory,ofs,((WORD2LONG*)&value)->wordlo);
   ofs=ofs+2;
 }
```

zuerst schreibt es das high word und dann das low word zu base_memory + ofs um ein DWORD zu speichern.

Die nächste C-Routine ist ein Beispiel um aus dem Dual Port RAM von 0 bis finalpos DWORDS zu lesen:

```
    ofs=0;
 for (ii=0; ii<(finalpos+1); ii++)
   {
   value1 = peek(base_memory,ofs);
     ofs=ofs+2;
     value2 = peek(base_memory,ofs);
     ofs=ofs+2;
   value=GetDWORD(value1,value2);
```

```
        printf("Value read: %lx\n",value);
        }
```

Die folgenden Typdefinitionen und Funktionen werden benötigt für obige Routinen:

```
typedef struct {
  unsigned short wordlo,wordhi;
} WORD2LONG;

//conversion of 2 WORD values to a DWORD

LONG GetDWORD(unsigned short val_1,unsigned short val_2)

{
  LONG temp;
  temp=0;

  ((WORD2LONG*)&temp)->wordhi=val_1;
  ((WORD2LONG*)&temp)->wordlo=val_2;

  return (temp);

}/*********************************/
```

Die obigen Routinen funktionieren nicht auf Windows-basierten Systemen da der Prozessor in Protected Mode arbeitet und daher keinen direkten IO und Speicherzugriff erlaubt. Daher sollten für Windows 98/ME und Windows 2000/XP die mitgelieferten Treiber und der DLL verwendet werden. Das GUI-Programm POSYS1800V4 verwendet die DLL und erlaubt die meisten Funktionen der POSYS® 1800 auszuprobieren.

Nach dem Einschalten ist der Dual Port RAM deaktiviert, sodass das BIOS es nicht sehen kann und ein mögliches Aufhängen des Rechners beim Booten verhindert wird. Das RAM kann in diesem Modus nur in Single Port RAM Modus verwendet werden. Alle Schreib- und Lesevorgänge werden durch den Motion Prozessor erledigt. Um den Dual Port RAM Modus zu aktivieren muss die Funktion

```
EnableDPRam(void)                              //DOS
EnableDPRam(hPosys:P1800_Handle);stdcall       //Windows DLL
```

verwendet werden.

Mit der Funktion

```
DisableDPRam(void)                             //DOS
DisableDPRam(hPosys:P1800_Handle);stdcall      //Windows DLL
```

kann der Dual Port RAM Modus jederzeit wieder deaktiviert werden.

Ein HardReset des Motion Prozessors deaktiviert auch den Dual Port RAM Modus. Die Funktion

```
EnableDPRam(void)                              //DOS
EnableDPRam(hPosys:P1800_Handle);stdcall       //Windows DLL
```

aktiviert wieder den Dual Port RAM Modus nach einem HardReset oder einem ISA-Bus Reset.

Für zusätzliche Informationen verweisen wir auf die weiteren Abschnitte bzgl. Speicherverwendung in diesem Handbuch.

Vorschläge zur Ermittlung geeigneter Speicherbereiche

Anmerkung: die untenstehenden Vorschläge zur Problembehebung gelten nur für Systeme basierend auf Betriebssysteme der Firma Microsoft, da diese Betriebssysteme, beginnend mit MSDOS bis zur Version XP, weiterhin exklusiv Speicherbereiche zwischen 640k und 1024k reservieren und damit die Verwendung von Dual Port RAM behindern.
Linux-basierte Betriebssysteme hingegen können ohne Bedenken verwendet werden, da sie diesen Speicherbereich flexibler verwenden. Bei der Verwendung unseres mitgelieferten Treibers für Linux (getestet mit SuSE 8.2, Red Hat 9.0 und Mandrake 8.2) konnten keine Einschränkungen festgestellt werden, wobei wir nicht ausschliessen können, dass mit anderen Versionen trotzdem Probleme auftreten können.
Ein weiterer Punkt den es zu beachten gibt, ist die Tatsache, dass nicht jeder PC gleichermaßen geeignet ist für den Einsatz mit Dual Port RAM-bestückten Karten, egal welches Betriebssystem installiert ist. Dieser sollte vor der endgültigen Wahl des einzusetzenden PCs auf seine Tauglichkeit zur Verwendung mit Dual Port RAM-bestückten ISA-Bus basierten Karten hin geprüft werden.

Jeder PC hat seine Eigenheiten speziell wenn es um die Verwendung des oberen Speicherbereiches (640k - 1MB) geht, was

BIOS bezogen ist. Aber dieser Bereich wird auch vom eingesetzten Betriebssystem mitbestimmt.

z.B.: der selbe PC kann bei Verwendung von MSDOS 6.22 mehr Speicherbereich zur Verfügung stellen, als wenn er mit DRDOS 7.03 betrieben wird, was den Wechsel des eingestellten Speicherbereiches auf der POSYS® 1800 erforderlich machen kann. Auf einem technisch anderen Computer kann es wiederum andersherum sein.

Obwohl MSD.EXE sehr wertvoll sein kann, freie Speicherbereiche zu bestimmen, ist es trotzdem nicht immer 100% zuverlässig und kann unter Umständen Bereiche anzeigen die an und für sich frei zu sein scheinen, aber trotzdem nicht verwendet können und anders herum kann es Bereiche anzeigen die belegt sind, aber trotzdem verwendet werden können. Nichtsdestotrotz kann es bei der Auswahl des verwendbaren Speicherbereiches erste hilfreiche Informationen liefern. Sollten als frei angezeigte Bereiche nicht funktionieren, dann raten wir dazu einfach andere, auch als belegt angezeigte Bereiche, zu testen.

Abhängig vom PC kann es auch notwendig sein den gewünschten Speicherbereich explizit bereits im BIOS zu reservieren. Manche PCs bieten diese Möglichkeit an, andere haben diese Funktion gar nicht implementiert, können aber trotzdem funktionieren, wiederum andere, die diese Funktion unterstützen, ignorieren sogar die Einstellung und funktionieren trotzdem nicht, egal welche Einstellung im BIOS gewählt wurde. Auch hier gilt wieder, dass Testen nur Gewissheit bringen kann.

Bei der Verwendung mehrerer Karten wird die Angelegenheit noch kritischer, da mit jeder zusätzlichen Karte auch zusätzlicher Speicherbereich reserviert werden muss. Hierdurch kann es dazu führen, dass der obere Speicherbereich sehr schnell zu klein wird. Eine Möglichkeit hier Abhilfe zu schaffen ist die POSYS® 1800 mit kleineren Dual Port RAM Modulen zu bestücken (z.B. 8k x 16 anstatt 16k x 16). Andere Größen sind selbstverständlich auch möglich. Auch die Möglichkeit bei wenig zur Verfügung stehenden Speicher bei mehreren mit größeren DPRAM Modulen bestückte POSYS® 1800 zu arbeiten, kann POSYS Motion Control GmbH & Co.KG eine flexible Lösung anbieten, ohne dass Komponenten ausgetauscht werden müssen. Bitte setzen Sie sich hierzu direkt mit uns in Verbindung.

Falls sie den Aufwand vermeiden wollen, verfügbare Speicherbereiche zu ermitteln, können wir die POSYS® 1800 auch mit Single Port RAM (512KB oder 1024KB) anbieten. Falls es gewünscht wird mehrere Karten im System einzusetzen kann es sein, dass dies die einzig praktikable Lösung für ein stabiles System ist. Mit Dual Port RAM bestückte Karten können aber auch im Modus "Single Port RAM" gelesen und beschrieben werden. Somit ist gegen deren Einsatz grundsätzlich nichts einzuwenden, ausser dass ein wesentlich geringerer Speicherbereich zur Verfügung gestellt wird, der zudem im Vergleich zu Single Port RAM einen nicht zu verachtenden Kostennachteil mit sich bringt.

Für weitere Informationen bei der Verwendung mit Dual Port RAM in Verbindung mit der POSYS® 1800 kontaktieren sie uns bitte direkt.

Date: 28.04.2005
Revision: 09.05.2005
E-mail: support@halbeck.com
POSYS® is a registered trade mark of POSYS Motion C ontrol

¤

POSYS® 9xx

Versionen 9xx

Liste der verfügbaren Versionen

POSYS® 90x für bürstenbehaftete Servomotoren

Die POSYS® 90x kann auch mit bürstenlosen Servomotoren eingesetzt werden vorausgesetzt entsprechende Verstärker werden verwendet.

Modell	# Achsen
POSYS® 902	2
POSYS® 904	4

POSYS® 95x für Schrittmotoren

Die POSYS® 95x kann auch für Mikroschrittmotoren verwendet werden wenn Mikroschritttreiber verwendet werden.

Modell	# Achsen
POSYS® 952	2
POSYS® 954	4

Alle Versionen sind auch für den erweiterten Temperaturbereich erhältlich (-40° - +85°C). Nicht auf Lager. Für Bestellungen fügen sie der Bestellnummer ein "-I" hinzu (z.B.: 904-I).

¤

Einstellbare Komponenten

Einstellbare Komponenten; POSYS® 9xx

In diesem Abschnitt erhalten Sie die Informationen die notwendig sind alle Komponenten korrekt mit einander zu verbinden. Das folgende Bild zeigt die Position der einzelnen Verbindungen auf mit ihren Bezeichnungen zur leichteren Orientierung.

Die wichtigsten anwender bezogenen Komponenten sind:

- Widerstandsnetzwerke RS1, RS2 und RS3. Sie bestimmen die Art der Enkoderterminierung
- Mode Jumper erlaubt die serielle Schnittstelle als primärer Kommunikationsport ausgewählt werden zu können. Ansonsten (Standard) ist es der PCI-Bus.
- Sync I/O um die Karte in den Master oder Slave Modus zu schalten, falls weitere Karten im System eingesetzt werden.

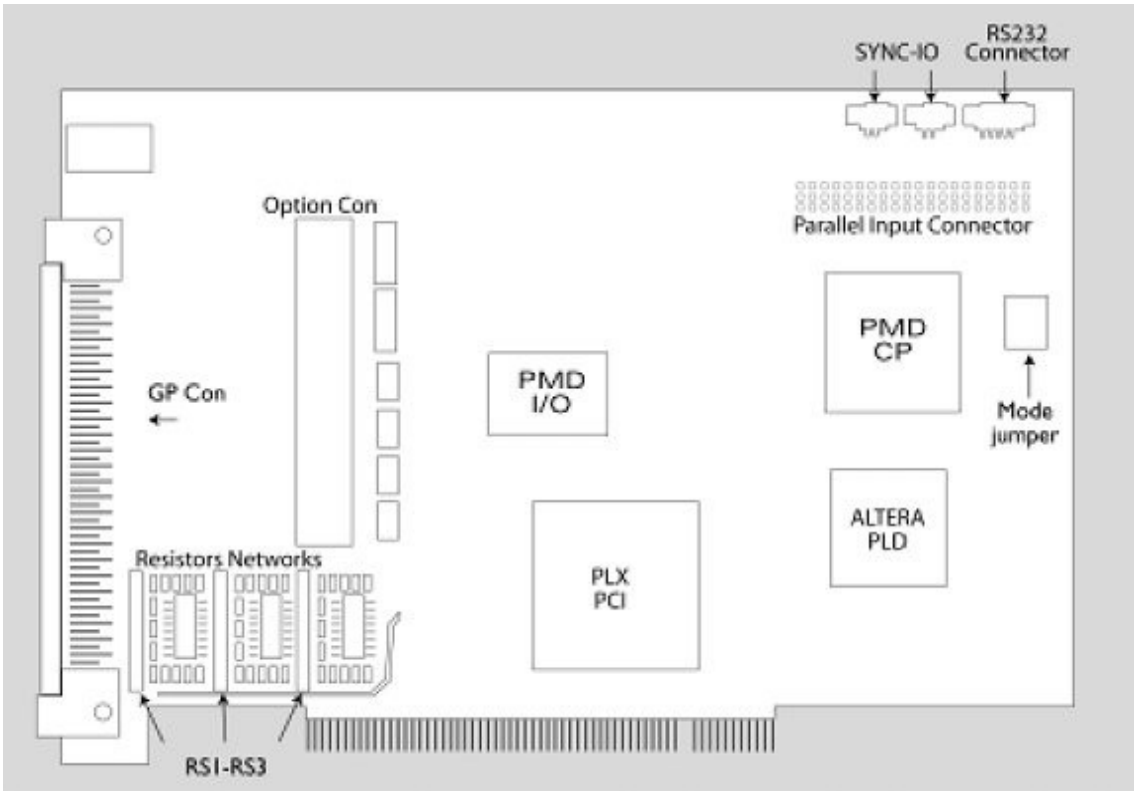


Abbildung zeigt POSYS® 9xx

Widerstandsnetzwerke

Die folgende Tabelle zeigt die Funktionen der Widerstandsnetzwerke RS1-RS3 auf:

Komponente	Stellung	Beschreibung
Netzwerk-widerstände	Installiert: RS1-RS3 (Standard)	Für die Verwendung von Enkodern mit invertierten Signalen sollten die Widerstandsnetzwerke installiert bleiben
	Deinstalliert: RS1-RS3	Für Enkoder mit TTL-Signalausgabe

Falls sowohl Enkoder mit TTL als auch Enkoder mit invertierten Signalen Verwendung finden, können Sie auf der Seite Resistor Networks for TTL or inverted Encoder Signals Detailinformationen finden wie die Widerstandsnetzwerke einzustellen sind.

Mode Jumper

Die folgende Tabelle zeigt die Funktionen des Mode Jumpers auf. Der Jumper ist nur auf den Karten POSYS 8xx-B und POSYS 9xx vorhanden. Die POSYS 7xx und 8xx verfügen nicht über diesen Jumper:

Komponente	Stellung	Beschreibung
Mode Jumper	1-2 (Standard)	Der PC-Bus ist in dieser Stellung als Hauptkommunikationsleitung definiert
	2-3	Der PC-Bus als Kommunikationsleitung ist deaktiviert, es ist nur noch möglich über die serielle Schnittstelle (RS232 = Standard; RS422/485 über optionale Module zusätzlich möglich) zu kommunizieren.

Parallel Input Connector "J1"

Pin No.	Beschreibung	Pin No.	Beschreibung
1	GND	2	A0
3	DS0	4	A1
5	DS1	6	A2
7	DS2	8	A3

Pin No.	Beschreibung	Pin No.	Beschreibung
9	DS3	10	A4
11	DS4	12	A5
13	DS5	14	A6
15	DS6	16	A7
17	DS7	18	A8
19	DS8	20	A9
21	DS9	22	A10
23	DS10	24	A11
25	DS11	26	A12
27	DS12	28	A13
29	DS13	30	A14
31	DS14	32	A15
33	DS15	34	IS-
35	DS-	36	STRB-
37	WE-	38	VCC
39	R/W	40	PS-

¤

Serielle Schnittstelle

Serial I/F Stecker PCI-Karte (POSYS 9xx)

Nur gültig für die POSYS 9xx Serie

Dieser Stecker stellt eine serielle Verbindung zur POSYS 9xx zur Verfügung. Der Serial I/F Stecker ist ein 5-poliger Molex MLX Micro-4-SMD Stecker

Pin	Verbindung	Beschreibung
1	SrlEnable	Serial enable (wird nur für RS422/485 benötigt)
2	SrlXmt	Serial transmit output
3	SrlRcv	Serial receive input
4	GND	Ground
5	Vcc	+5V

Diese Schnittstelle erlaubt die POSYS 9xx über die serielle Schnittstelle RS232 zu betreiben. Sie kann als primäre Kommunikationsschnittstelle eingestellt werden und erlaubt die Verwendung aller Kommandos oder als sekundäre Schnittstelle über die nur GET-Kommandos für Überwachungszwecke akzeptiert werden.

Das serielle Kabel ist optional. Die folgende Beschreibung listet die notwendigen Komponenten auf falls es gewünscht ist ein eigenes serielles Kabel zu verwenden.
5-poliges Molex MicroBlade SMD Stecker auf der Karte (Molex Bestellnummer: 53398-0571)
Gegenstück für Kabel Molex Bestellnummer: 51021-0500

Die Standardeinstellungen für die serielle Kommunikation nach dem Einschalten sind:
Geschwindigkeit: 9600 Baud/sek.
Parität: keine
Stop Bit: 1
Protokoll: point-to-point

Falls andere Standardeinstellungen gewünscht werden, dann setzen Sie sich bitte mit POSYS Motion Control GmbH & Co.KG in Verbindung. Diese Werte sind aus kompatibilitätsgründen gewählt worden, können aber kundenspezifisch angepasst werden. Beim Einschalten würden diese neuen Werte sofort effektiv.

Set/GetSerialPortMode, Serial Port Description

¤

Bürstenbehaftete Servomotor Pinout Beschreibung

Pinout Beschreibung für bürstenbehaftete oder bürstenlose (mit externe Kommutierung) Servomotoren

POSYS® 90x

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
1	A1+	Channel A+ encoder signal (X)	51	A3+	Channel A+ encoder signal (Z)
2	A1-	Channel A- inverted encoder signal (X)	52	A3-	Channel A- inverted encoder signal (Z)
3	B1+	Channel B+ encoder signal (X)	53	B3+	Channel B+ encoder signal (Z)
4	B1-	Channel B- inverted encoder signal (X)	54	B3-	Channel B- inverted encoder signal (Z)
5	I1+	Channel I+ encoder signal (X)	55	I3+	Channel I+ encoder signal (Z)
6	I1-	Channel I- inverted encoder signal (X)	56	I3-	Channel I- inverted encoder signal (Z)
7	Vc1	+5V	57	Vc1	+5V
8	GND	Ground	58	GND	Ground
9	XPLIM1	Limit switch positive direction (X)	59	XPLIM3	Limit switch positive direction (Z)
10	XNLIM1	Limit switch negative direction (X)	60	XNLIM3	Limit switch negative direction (Z)
11	HOME1	Home signal (X) or high speed latch	61	HOME3	Home signal (Z) or high speed latch
12	GND	Ground	62	GND	Ground
13	OUT1	User programmable output (X)	63	OUT3	User programmable output (Z)
14	BPWM1	PWM magnitude (X)	64	BPWM3	PWM magnitude (Z)
15	BPWMS1	PWM sign (X)	65	BPWMS3	PWM sign (Z)
16	AXIN1	User programmable input (X)	66	AXIN3	User programmable input (Z)
17	DACV1	Motor command (X), ±10V signal	67	DACV3	Motor command (Z), ±10V signal
18	AGND	Ground for motor command	68	AGND	Ground for motor command
19	A2+	Channel A+ encoder signal (Y)	69	A4+	Channel A+ encoder signal (W)
20	A2-	Channel A- inverted encoder signal (Y)	70	A4-	Channel A- inverted encoder signal (W)
21	B2+	Channel B+ encoder signal (Y)	71	B4+	Channel B+ encoder signal (W)
22	B2-	Channel B- inverted encoder signal (Y)	72	B4-	Channel B- inverted encoder signal (W)
23	I2+	Channel I+ encoder signal (Y)	73	I4+	Channel I+ encoder signal (W)
24	I2-	Channel I- inverted encoder signal (Y)	74	I4-	Channel I- inverted encoder signal (W)
25	Vc1	+5V	75	Vc1	+5V
26	GND	Ground	76	GND	Ground
27	XPLIM2	Limit switch positive direction (Y)	77	XPLIM4	Limit switch positive direction (W)
28	XNLIM2	Limit switch negative direction (Y)	78	XNLIM4	Limit switch negative direction (W)
29	HOME2	Home signal (Y) or high speed latch	79	HOME4	Home signal (W) or high speed latch
30	OUT2	User programmable output (Y)	80	OUT4	User programmable output (W)
31	BPWM2	PWM magnitude (Y)	81	BPWM4	PWM magnitude (W)
32	BPWMS2	PWM sign (Y)	82	BPWMS4	PWM sign (W)

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
33	AXIN2	User programmable input (Y)	83	AXIN4	User programmable input (W)
34	DACV2	Motor command (Y), ±10V signal	84	DACV4	Motor command (W), ±10V signal
35	AGND	Ground for motor command	85	AGND	Ground for motor command
36	UI0	uncommitted input (0)	86	UI4	uncommitted input (4)
37	UI1	uncommitted input (1)	87	UI5	uncommitted input (5)
38	UI2	uncommitted input (2)	88	UI6	uncommitted input (6)
39	UI3	uncommitted input (3)	89	UI7	uncommitted input (7)
40	AMPEN1	Amplifier enable (X) or user programmable output	90	AMPEN3	Amplifier enable (Z) or user programmable output
41	UO0	uncommitted output (0)	91	UO4	uncommitted output (4)
42	UO1	uncommitted output (1)	92	UO5	uncommitted output (5)
43	UO2	uncommitted output (2)	93	UO6	uncommitted output (6)
44	UO3	uncommitted output (3)	94	UO7	uncommitted output (7)
45	AMPEN2	Amplifier enable (Y) or user programmable output	95	AMPEN4	Amplifier enable (W) or user programmable output
46	RS-	Hardware reset	96	ANGND	GND for analog inputs
47	Ainput1	analog input channel (0)	97	Ainput5	analog input channel (4)
48	Ainput2	analog input channel (1)	98	Ainput6	analog input channel (5)
49	Ainput3	analog input channel (2)	99	Ainput7	analog input channel (6)
50	Ainput4	analog input channel (3)	100	Ainput8	analog input channel (7)

¤

Schrittmotor Pinout Beschreibung

Pinout Beschreibung Schritt-/Mikroschrittmotor* Version

POSYS® 95x

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
1	A1+	Channel A+ encoder signal (X)	51	A3+	Channel A+ encoder signal (Z)
2	A1-	Channel A- inverted encoder signal (X)	52	A3-	Channel A- inverted encoder signal (Z)
3	B1+	Channel B+ encoder signal (X)	53	B3+	Channel B+ encoder signal (Z)
4	B1-	Channel B- inverted encoder signal (X)	54	B3-	Channel B- inverted encoder signal (Z)
5	I1+	Channel I+ encoder signal (X)	55	I3+	Channel I+ encoder signal (Z)
6	I1-	Channel I- inverted encoder signal (X)	56	I3-	Channel I- inverted encoder signal (Z)
7	Vc1	+5V	57	Vc1	+5V
8	GND	Ground	58	GND	Ground
9	XPLIM1	Limit switch positive direction (X)	59	XPLIM3	Limit switch positive direction (Z)
10	XNLIM1	Limit switch negative direction (X)	60	XNLIM3	Limit switch negative direction (Z)
11	HOME1	Home signal (X) or high speed latch	61	HOME3	Home signal (Z) or high speed latch
12	GND	Ground	62	GND	Ground
13	AXISOUT1	User programmable output (X)	63	AXISOUT3	User programmable output (Z)
14	PULSE1	Pulse (X)	64	PULSE3	Pulse (Z)
15	DIR1	Direction (X)	65	DIR3	Direction (Z)
16	AXIN1	User programmable input (X)	66	AXIN3	User programmable input (Z)
17	ATREST1	At Rest (X)	67	ATREST3	At Rest (Z)

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
18	GND	Ground	68	GND	Ground
19	A2+	Channel A+ encoder signal (Y)	69	A4+	Channel A+ encoder signal (W)
20	A2-	Channel A- inverted encoder signal (Y)	70	A4-	Channel A- inverted encoder signal (W)
21	B2+	Channel B+ encoder signal (Y)	71	B4+	Channel B+ encoder signal (W)
22	B2-	Channel B- inverted encoder signal (Y)	72	B4-	Channel B- inverted encoder signal (W)
23	I2+	Channel I+ encoder signal (Y)	73	I4+	Channel I+ encoder signal (W)
24	I2-	Channel I- inverted encoder signal (Y)	74	I4-	Channel I- inverted encoder signal (W)
25	Vc1	+5V	75	Vc1	+5V
26	GND	Ground	76	GND	Ground
27	XPLIM2	Limit switch positive direction (Y)	77	XPLIM4	Limit switch positive direction (W)
28	XNLIM2	Limit switch negative direction (Y)	78	XNLIM4	Limit switch negative direction (W)
29	HOME2	Home signal (Y) or high speed latch	79	HOME4	Home signal (W) or high speed latch
30	AXISOUT2	User programmable output (Y)	80	AXISOUT4	User programmable output (W)
31	PULSE2	Pulse (Y)	81	PULSE4	Pulse (W)
32	DIR2	Direction (Y)	82	DIR4	Direction (W)
33	AXIN2	User programmable input (Y)	83	AXIN4	User programmable input (W)
34	ATREST2	At Rest (Y)	84	ATREST4	At Rest (W)
35	GND	Ground	85	GND	Ground
36	UI0	uncommitted input (0)	86	UI4	uncommitted input (4)
37	UI1	uncommitted input (1)	87	UI5	uncommitted input (5)
38	UI2	uncommitted input (2)	88	UI6	uncommitted input (6)
39	UI3	uncommitted input (3)	89	UI7	uncommitted input (7)
40	AMPEN1	Amplifier enable (X)	90	AMPEN3	Amplifier enable (Z)
41	UO0	uncommitted output (0)	91	UO4	uncommitted output (4)
42	UO1	uncommitted output (1)	92	UO5	uncommitted output (5)
43	UO2	uncommitted output (2)	93	UO6	uncommitted output (6)
44	UO3	uncommitted output (3)	94	UO7	uncommitted output (7)
45	AMPEN2	Amplifier enable (Y)	95	AMPEN4	Amplifier enable (W)
46	RS-	Hardware reset	96	ANGND	GND for analog inputs
47	Ainput1	analog input channel (0)	97	Ainput5	analog input channel (4)
48	Ainput2	analog input channel (1)	98	Ainput6	analog input channel (5)
49	Ainput3	analog input channel (2)	99	Ainput7	analog input channel (6)
50	Ainput4	analog input channel (3)	100	Ainput8	analog input channel (7)

*Mikroschritt mit geeignetem Treiber

Pinout Beschreibung Schrittmotor Version; Option Connector POSYS® 95x

POSYS® 95x (68-position SCSI)

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
1	n.c.	No Connection	2	n.c.	No Connection
3	n.c.	No Connection	4	GND	Ground
5	n.c.	No Connection	6	GND	Ground
7	n.c.	No Connection	8	n.c.	No Connection
9	n.c.	No Connection	10	GND	Ground
11	n.c.	No Connection	12	GND	Ground

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
13	n.c.	No Connection	14	n.c.	No Connection
15	n.c.	No Connection	16	GND	Ground
17	n.c.	No Connection	18	GND	Ground
19	n.c.	No Connection	20	n.c.	No Connection
21	n.c.	No Connection	22	GND	Ground
23	n.c.	No Connection	24	GND	Ground
25	n.c.	No Connection	26	n.c.	No Connection
27	n.c.	No Connection	28	GND	Ground
29	n.c.	No Connection	30	n.c.	No Connection
31	n.c.	No Connection	32	GND	Ground
33	n.c.	No Connection	34	n.c.	No Connection
35	n.c.	No Connection	36	GND	Ground
37	n.c.	No Connection	38	n.c.	No Connection
39	n.c.	No Connection	40	GND	Ground
41	Pulse1+	Pulse+ (X)	42	Pulse1-	Pulse- (X)
43	Pulse2+	Pulse+ (Y)	44	Pulse2-	Pulse- (Y)
45	Pulse3+	Pulse+ (Z)	46	Pulse3-	Pulse- (Z)
47	Pulse4+	Pulse+ (W)	48	Pulse4-	Pulse- (W)
49	Direction1+	Direction+ (X)	50	Direction1-	Direction- (X)
51	Direction2+	Direction+ (Y)	52	Direction2-	Direction- (Y)
53	Direction3+	Direction+ (Z)	54	Direction3-	Direction- (Z)
55	Direction4+	Direction+ (W)	56	Direction4-	Direction+ (W)
57	Vc1	+5V	58	Vc1	+5V
59	GND	Ground	60	GND	Ground
61	n.c.	No Connection	62	AGND	No Connection
63	n.c.	No Connection	64	AGND	No Connection
65	n.c.	No Connection	66	AGND	No Connection
67	n.c.	No Connection	68	AGND	No Connection

¤

Analog Eingang Pinout Beschreibung

Analog Eingang Pinout Beschreibung (nur gültig für POSYS® 900)

J7 Pin-Out	Beschreibung	Erklärung
47	Analog Input 1	Analog input für channel (0)
48	Analog Input 2	Analog input für channel (1)
49	Analog Input 3	Analog input für channel (2)
50	Analog Input 4	Analog input für channel (3)

J1 Pin-Out	Beschreibung	Erklärung
97	Analog Input 5	Analog input für channel (4)
98	Analog Input 6	Analog input für channel (5)
99	Analog Input 7	Analog input für channel (6)
100	Analog Input 8	Analog input für channel (7)

Analog1-8 sind 8 frei-programmierbare analoge Eingangssignale. Falls angeschlossen stellen die Spannungen an den Eingängen zwar noch nichts dar und der Motion Prozessor reagiert nicht darauf, allerdings stellen sie eine gute Möglichkeit

dar, dass des Anwender's Programm interaktiv hierrauf reagiert. Diese Signale werden mit dem Kommando ReadAnalog gelesen.

Die minimal erlaubte Eingangsspannung beträgt 0,0V und die maximale erlaubte Eingangsspannung beträgt 4.096V. Die Auflösung beträgt 10 Bit. Um den numerischen Wert zu bestimmen kann folgende Formel verwendet werden:

ReadValue(Channel) = (AnalogVoltage * 65,536) / 4.096V

Umgekehrt wird mit folgender Formel die entsprechende Spannung berechnet:

AnalogVoltage = (ReadValue(Channel) * 4.096V) / 65,536

Damit die Analogspannungen korrekt ermittelt werden können, müssen die Analogsignale zusätzlich mit AnalogGND (Analog Ground) verbunden sein.

Falls andere als die standardmäßigen Eingangsspannungen gewünscht sind, können nach Absprache folgende Spannungen ersatzweise und optional angeboten werden: 1.8V, 2.5V, 3.0V und 3.3V.

¤

POSYS® 19xx

Versionen 19xx

Liste der verfügbaren Versionen

POSYS® 192x für bürstenbehaftete/bürstenlose (interne Kommutierung) Servomotoren, Mikroschritt und Schrittmotoren.

Die POSYS® 192x kann für alle Motortypen verwendet werden (bürstenbehaftete Servomotoren, bürstenlose Servomotoren (interne und externe Kommutierung, Mikroschrittmotoren und Schrittmotoren)

Modell	# Achsen
POSYS® 1922	2
POSYS® 1924	4

POSYS® 195x für Schrittmotoren

Die POSYS® 195x kann auch für Mikroschrittmotoren verwendet werden wenn Mikroschritttreiber verwendet werden.

Modell	# Achsen
POSYS® 1952	2
POSYS® 1954	4

Alle Versionen sind auch für den erweiterten Temperaturbereich erhältlich (-40° - +85°C). Nicht auf Lager. Für Bestellungen fügen sie der Bestellnummer ein "-I" hinzu (z.B.: 1904-I).

¤

Einstellbare Komponenten

Einstellbare Komponenten; POSYS® 19xx

In diesem Abschnitt erhalten Sie die Informationen die notwendig sind alle Komponenten korrekt mit einander zu verbinden. Das folgende Bild zeigt die Position der einzelnen Verbindungen auf mit ihren Bezeichnungen zur leichteren Orientierung.

Die wichtigsten anwender bezogenen Komponenten sind:

- WiderstandsnetzwerkeRS1, RS2 und RS3. Sie bestimmen die Art der Enkoderterminierung
- Mode Jumper erlaubt die serielle Schnittstelle als primärer Kommunikationsport ausgewählt werden zu können. Ansonsten (Standard) ist es der PCI-Bus.
- Sync I/O um die Karte in den Master oder Slave Modus zu schalten, falls weitere Karten im System eingesetzt werden.
- CAN 2.0B kann als Kommunikationsport selektiert werden, falls dies den gewünschten Kommunikationsmodus darstellt.
- JP1 und JP2 müssen selektiert sein (entweder/oder) um die jeweilige Achse für den Servo- oder Schrittmotorbetrieb verwenden zu können (nicht 195x).

Im Folgenden werden diese Einstellungen im Detail besprochen:

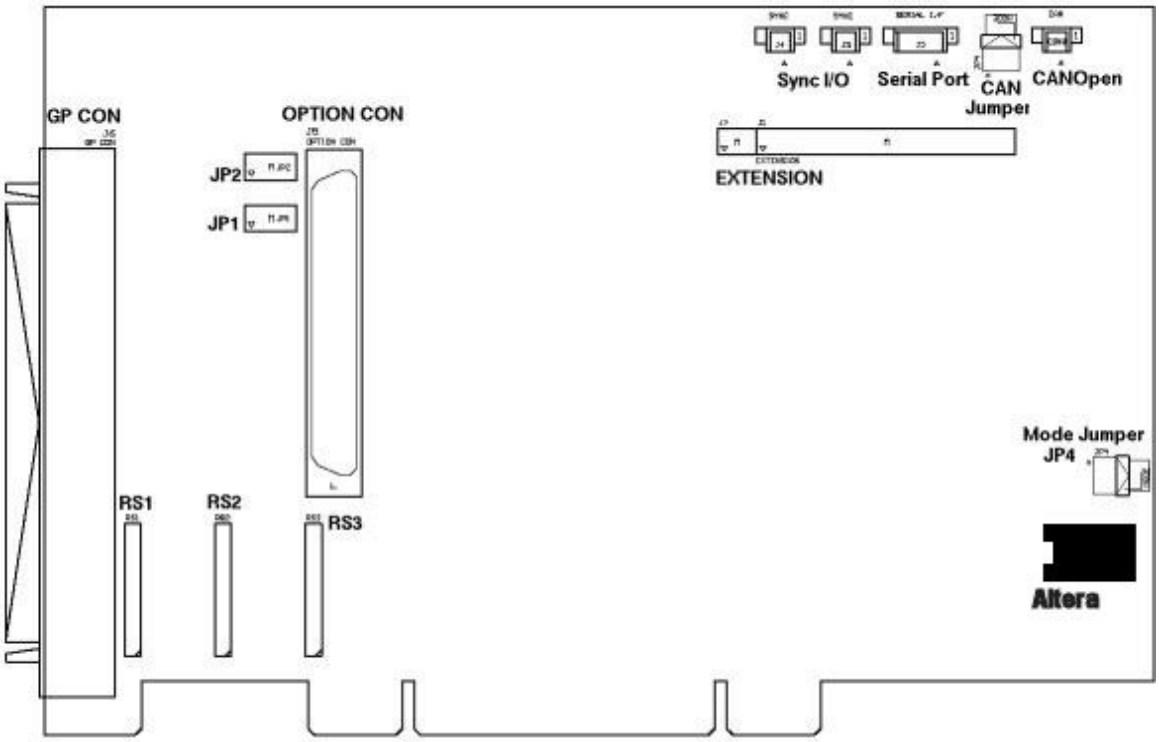


Abbildung zeigt POSYS® 19xx

Widerstandsnetzwerke RS1, RS2, RS3

Die folgende Tabelle zeigt die Funktionen der Widerstandsnetzwerke RS1-RS3 auf:

Komponente	Stellung	Beschreibung
Netzwerk-widerstände	Installiert: RS1-RS3 (Standard)	Für die Verwendung von Enkodern mit invertierten Signalen sollten die Widerstandsnetzwerke installiert bleiben
	Deinstalliert: RS1-RS3	Für Enkoder mit TTL-Signalausgabe
	Sollten beide Versionen von Enkodern im System eingesetzt werden, ist es auch möglich die Widerstandsnetzwerke entsprechend anzupassen. Bitte setzen Sie sich mit uns in Verbindung für die genauen Pinpositionen der Widerstandsnetzwerke	

Falls sowohl Enkoder mit TTL als auch Enkoder mit invertierten Signalen Verwendung finden, können Sie auf der Seite Resistor Networks for TTL or inverted Encoder Signals Detailinformationen finden wie die Widerstandsnetzwerke einzustellen sind.

Mode Jumper „JP4“

Die folgende Tabelle zeigt die Funktionen des Mode Jumpers auf:

Komponente	Stellung	Beschreibung
Mode Jumper	1-2 (Standard)	Der PC-Bus ist in dieser Stellung als Hauptkommunikationsleitung definiert. Die serielle Schnittstelle ist alternativ als Kommunikationsport ohne Einschränkung verwendbar.
	2-3	Der PCI-Bus als Kommunikationsleitung ist deaktiviert, es ist nur noch möglich über die serielle Schnittstelle (RS232 = Standard; RS422/485 über optionale Module zusätzlich möglich) zu kommunizieren.

¤

SYNC IO

SYNC I/O

Diese beiden Stecker erlauben es mehrere im System befindliche Karten mit einander zu synchronisieren, sodass sie innerhalb des gleichen Arbeitszyklus arbeiten. Falls diese Verbindung nicht verwendet wird, fangen die anderen Karten erst mit ihrer Arbeit an nachdem die 1. Karte initialisiert wurde. Dies kann 256 µs nach einer 4-Achsen Karte sein. Mit aktiviertem SYNC I/O würden alle zusätzlichen Karten innerhalb von 50 ns nach dem Master anfangen zu arbeiten. Dies erlaubt präzises synchronisieren aller implementierten Achsen. Das Kommando Set/GetSynchronizationMode erklärt die Programmierung dieses Modus. Zusätzlich benötigen Sie das optionale SYNC I/O Kabel (1 für jede Kombination von 2 Karten).

Die Steckverbindung auf der Karte ist ein Molex Stecker (Artikelnummer: 53261-0271). Das dazu gehörige Gegenstück ist eine Molex Buchse (Artikelnummer: Molex 51021-0200, Crimpkontakt: Molex 50079-8100).

Set/GetSynchronizationMode, Sync IO Description

¤

Serielle Schnittstelle

Serial I/F Stecker PCI-Karte (POSYS 19xx)

Nur gültig für die POSYS 19xx Serie

Dieser Stecker stellt eine serielle Verbindung zur POSYS 19xx zur Verfügung. Der Serial I/F Stecker ist ein 5-poliger Molex MLX Micro-4-SMD Stecker

Pin	Verbindung	Beschreibung
1	SrlEnable	Serial enable (wird nur für RS422/485 benötigt)
2	SrlXmt	Serial transmit output
3	SrlRcv	Serial receive input
4	GND	Ground
5	Vcc	+5V

Diese Schnittstelle erlaubt die POSYS 19xx über die serielle Schnittstelle RS232 zu betreiben. Sie kann als primäre oder sekundäre (z.B für Monitoringaufgaben) Kommunikationsschnittstelle verwendet werden.

Das serielle Kabel ist optional. Die folgende Beschreibung listet die notwendigen Komponenten auf falls es gewünscht ist ein eigenes serielles Kabel zu verwenden.
5-poliges Molex MicroBlade SMD Stecker auf der Karte (Molex Bestellnummer: 53398-0571)
Gegenstück für Kabel Molex Bestellnummer: 51021-0500

Die Standardeinstellungen für die serielle Kommunikation nach dem Einschalten sind:
Geschwindigkeit: 57200 Baud/sek.
Parität: keine
Stop Bit: 2
Protokoll: point-to-point

Falls andere Standardeinstellungen gewünscht werden, dann setzen Sie sich bitte mit POSYS Motion Control GmbH & Co.KG in Verbindung. Diese Werte sind aus kompatibilitätsgründen gewählt worden, können aber kundenspezifisch angepasst werden. Beim Einschalten würden diese neuen Werte sofort effektiv.

Set/GetSerialPortMode, Serial Port Description

¤

CAN 2.0B

CAN 2.0B

Diese Verbindung erlaubt die Kommunikation mit der POSYS® via CAN 2.0B. In diesem Modus kann die POSYS® 1900/1800 auch als Standalone Controller verwendet. Es werden nur 5V Spannungsversorgung benötigt. Um die POSYS® in CAN 2.0B Modus als Standalone Controller verwenden zu können, können wir spezielle Micro Boxes anbieten, welche, je nach Modell, PCI- bzw. ISA-Slots bieten um die Karten mit der erfordelichen Spannung zu versorgen. Zusätzliche CPU-Karten sind nicht notwendig.

Diese Methode erlaubt es bis zu 128 „Nodes“ (Knoten) zu verbinden und somit bis zu 512 Achsen zu kombinieren.

Feldbusse erlauben leider in dem Maße keine Hochgeschwindigkeitskommunikation für präzise Interpolation. Die Kombination mehrerer Achsen auf einer Karte welches via CAN 2.0B mit einem Host kommuniziert, erlaubt aber nun genau dies – präzise Interpolation (linear und zirkular (Option)) auf einem Feldbus.

Das CAN 2.0B Kabel ist optional. Bitte kontaktieren Sie uns falls Sie es wünschen die POSYS® im CAN Modus zu betreiben.

Die Steckverbindung auf der Karte ist ein Molex Stecker (Artikelnummer: 53261–0271). Das dazu gehörige Gegenstück ist eine Molex Buchse (Artikelnummer: Molex 51021-0200, Crimpkontakt: Molex 50079-8100).

Für weitere Information verweisen wir auch auf folgende Seiten: CAN (Controller Area Network, Konfiguration der CAN2.0B Schnittstelle, CAN Ereignis Benachrichtigung and Set/GetCanMode)

¤

Motor Mode Konfiguration; POSYS® 19xx

Motor Mode Jumpers „JP1“ und „JP2“

Je nach Modell gibt es verschiedene Möglichkeiten die Achsen der POSYS® für die Verwendung mit bestimmten Motortypen, wie z.B. bürstenbehaftete/bürstenlose (mit und ohne On-Board Kommutierung) Servomotoren, Mikroschritt oder Schrittmotoren zu verwenden. Neben dem Hardwaresetup für verschiedene Motortypen, sind auch verschiedene Softwareeinstellungen vor zu nehmen. Die Softwareeinstellungen sind im Programmierhandbuch beschrieben. Die Standardeinstellung für jede Achse nach dem Einschalten ist generell „bürstenbehafteter Servomotor“, ausser für die Schrittmotorversion der Karte. Dann ist generell „Schrittmotor“ eingestellt. Falls Sie eine andere Standardeinstellung nach dem Einschalten wünschen, dann teilen Sie uns das bitte mit (z.B.: 2 Achsen = DC Brushless, 1 Achse = Schritt, 1 Achse = DC Bürstenbehaftet). Diese Einstellungen können fix auf der Karte kodiert werden.

Die folgende Sektion beschreibt nur die Hardwarekonfiguration für den einen oder anderen Motortyp, eine gemischte Stellung ist möglich. Bedenken Sie aber, dass dann die jeweilige Achse für die eine oder andere Methode konfiguriert ist. Um dies wieder zu ändern, müssen die Steckbrücken JP1 und JP2 erneut ihren Erfordernissen angepasst werden:

Motortyp	Jumper Stellung
DC Brushed	JP2
DC Brushless	JP2
Mikroschritt	JP2
Schritt	JP1

JP1 und JP2 bestehen beide aus 4 Steckbrücken. Jede Steckbrücke steht für Achse 1, 2, 3 oder 4 und ist auch abhängig davon ob Sie eine 1, 2, 3 oder 4 Achsen Karte erworben haben. Auch hängen die Möglichkeiten von der Kartenversion ab. Die folgende Tabelle beschreibt welche Steckbrückenposition notwendig ist um das korrekte Motorausgangssignal für den angeschlossenen Verstärker/Motor zu erhalten.

POSYS® 192x

Die POSYS® 192x kann für die Verwendung mit DC bürstenbehafteten Servomotoren, bürstenlosen Servomotoren mit interner/externer Kommutierung, Mikroschritt oder Schrittmotor eingestellt werden. Die folgende Tabelle beschreibt die korrekte Positionierung der Steckbrücken JP1 und JP2 für die eine oder andere Motorversion:

Achse	Motortyp	
	DC Bürstenbehafteter Servomotor DC Bürstenloser Servomotor (m/ interner/externer Kommutierung) oder Mikroschrittmotor	Schritt- oder Mikroschrittmotor (m/ Mikroschritttreiber)
1	JP2 kurz (erste von links); JP1 offen	JP1 kurz (erste von links); JP2 offen
2	JP2 kurz (zweite von links); JP1 offen	JP1 kurz (zweite von links); JP2 offen

3	JP2 kurz (dritte von links); JP1 offen	JP1 kurz (dritte von links); JP2 offen
4	JP2 kurz (vierte von links); JP1 offen	JP1 kurz (vierte von links); JP2 offen

POSYS® 195x

Die POSYS® 195x kann nur für die Verwendung mit Schrittmotoren eingerichtet werden. Die Verwendung von Servomotorverstärker ist aber trotzdem dann möglich. Es gibt einige Hersteller (Yaskawa, Mitsubishi, Panasonic, Tamagawa usw.) die Verstärker mit interner PI- oder PID-Regelung herstellen und das Takt- & Richtungssignal der POSYS® in entsprechende Bewegung umsetzen. Bei Verwendung des entsprechenden Treibers ist auch der Einsatz von Mikroschrittmotoren möglich. Die folgende Tabelle beschreibt die korrekte Positionierung der Steckbrücken JP1 und JP2 für die eine oder andere Motorversion:

Achse	Motortyp	
	DC Bürstenbehafteter Servomotor DC Bürstenloser Servomotor (mit interner/externer Kommutierung) oder Mikroschrittmotor	Schritt- oder Mikroschrittmotor (mit Mikroschritttreiber)
1	n/a	JP1 kurz (erste von links); JP2 offen
2	n/a	JP1 kurz (zweite von links); JP2 offen
3	n/a	JP1 kurz (dritte von links); JP2 offen
4	n/a	JP1 kurz (vierte von links); JP2 offen

¤

Bürstenbehaftete Servomotor Pinout Beschreibung

Pinout Beschreibung für bürstenbehaftete oder bürstenlose (mit externe Kommutierung) Servomotoren

POSYS® 190x/192x

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
1	A1+	Channel A+ encoder signal (X)	51	A3+	Channel A+ encoder signal (Z)
2	A1-	Channel A- inverted encoder signal (X)	52	A3-	Channel A- inverted encoder signal (Z)
3	B1+	Channel B+ encoder signal (X)	53	B3+	Channel B+ encoder signal (Z)
4	B1-	Channel B- inverted encoder signal (X)	54	B3-	Channel B- inverted encoder signal (Z)
5	I1+	Channel I+ encoder signal (X)	55	I3+	Channel I+ encoder signal (Z)
6	I1-	Channel I- inverted encoder signal (X)	56	I3-	Channel I- inverted encoder signal (Z)
7	Vc1	+5V	57	Vc1	+5V
8	GND	Ground	58	GND	Ground
9	XPLIM1	Limit switch positive direction (X)	59	XPLIM3	Limit switch positive direction (Z)
10	XNLIM1	Limit switch negative direction (X)	60	XNLIM3	Limit switch negative direction (Z)
11	HOME1	Home signal (X) or high speed latch	61	HOME3	Home signal (Z) or high speed latch
12	GND	Ground	62	GND	Ground
13	OUT1	User programmable output (X)	63	OUT3	User programmable output (Z)
14	BPWM1	PWM magnitude (X)	64	BPWM3	PWM magnitude (Z)
15	BPWMS1	PWM sign (X)	65	BPWMS3	PWM sign (Z)
16	AXIN1	User programmable input (X)	66	AXIN3	User programmable input (Z)
17	DACV1	Motor command (X), ±10V signal	67	DACV3	Motor command (Z), ±10V signal
18	AGND	Ground for motor command	68	AGND	Ground for motor command

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
19	A2+	Channel A+ encoder signal (Y)	69	A4+	Channel A+ encoder signal (W)
20	A2-	Channel A- inverted encoder signal (Y)	70	A4-	Channel A- inverted encoder signal (W)
21	B2+	Channel B+ encoder signal (Y)	71	B4+	Channel B+ encoder signal (W)
22	B2-	Channel B- inverted encoder signal (Y)	72	B4-	Channel B- inverted encoder signal (W)
23	I2+	Channel I+ encoder signal (Y)	73	I4+	Channel I+ encoder signal (W)
24	I2-	Channel I- inverted encoder signal (Y)	74	I4-	Channel I- inverted encoder signal (W)
25	Vc1	+5V	75	Vc1	+5V
26	GND	Ground	76	GND	Ground
27	XPLIM2	Limit switch positive direction (Y)	77	XPLIM4	Limit switch positive direction (W)
28	XNLIM2	Limit switch negative direction (Y)	78	XNLIM4	Limit switch negative direction (W)
29	HOME2	Home signal (Y) or high speed latch	79	HOME4	Home signal (W) or high speed latch
30	OUT2	User programmable output (Y)	80	OUT4	User programmable output (W)
31	BPWM2	PWM magnitude (Y)	81	BPWM4	PWM magnitude (W)
32	BPWMS2	PWM sign (Y)	82	BPWMS4	PWM sign (W)
33	AXIN2	User programmable input (Y)	83	AXIN4	User programmable input (W)
34	DACV2	Motor command (Y), ±10V signal	84	DACV4	Motor command (W), ±10V signal
35	AGND	Ground for motor command	85	AGND	Ground for motor command
36	UI0	uncommitted input (0)	86	UI4	uncommitted input (4)
37	UI1	uncommitted input (1)	87	UI5	uncommitted input (5)
38	UI2	uncommitted input (2)	88	UI6	uncommitted input (6)
39	UI3	uncommitted input (3)	89	UI7	uncommitted input (7)
40	AMPEN1	Amplifier enable (X) or user programmable output	90	AMPEN3	Amplifier enable (Z) or user programmable output
41	UO0	uncommitted output (0)	91	UO4	uncommitted output (4)
42	UO1	uncommitted output (1)	92	UO5	uncommitted output (5)
43	UO2	uncommitted output (2)	93	UO6	uncommitted output (6)
44	UO3	uncommitted output (3)	94	UO7	uncommitted output (7)
45	AMPEN2	Amplifier enable (Y) or user programmable output	95	AMPEN4	Amplifier enable (W) or user programmable output
46	RS-	Hardware reset	96	ANGND	GND for analog inputs
47	Ainput1	analog input channel (0)	97	Ainput5	analog input channel (4)
48	Ainput2	analog input channel (1)	98	Ainput6	analog input channel (5)
49	Ainput3	analog input channel (2)	99	Ainput7	analog input channel (6)
50	Ainput4	analog input channel (3)	100	Ainput8	analog input channel (7)

¤

Bürstenlose Servo- und Mikroschrittmotoren Pinout Beschreibung

Pinout Beschreibung bürstenlose Servomotoren mit interne Kommutierung, Mikroschritt und Multiple Motor Modus

POSYS® 192x

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
1	A1+	Channel A+ encoder signal (X)	51	A3+	Channel A+ encoder signal (Z)

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
2	A1-	Channel A- inverted encoder signal (X)	52	A3-	Channel A- inverted encoder signal (Z)
3	B1+	Channel B+ encoder signal (X)	53	B3+	Channel B+ encoder signal (Z)
4	B1-	Channel B- inverted encoder signal (X)	54	B3-	Channel B- inverted encoder signal (Z)
5	I1+	Channel I+ encoder signal (X)	55	I3+	Channel I+ encoder signal (Z)
6	I1-	Channel I- inverted encoder signal (X)	56	I3-	Channel I- inverted encoder signal (Z)
7	Vc1	+5V	57	Vc1	+5V
8	GND	Ground	58	GND	Ground
9	XPLIM1	Limit switch positive direction (X)	59	XPLIM3	Limit switch positive direction (Z)
10	XNLIM1	Limit switch negative direction (X)	60	XNLIM3	Limit switch negative direction (Z)
11	HOME1	Home signal (X) or high speed latch	61	HOME3	Home signal (Z) or high speed latch
12	GND	Ground	62	GND	Ground
13	OUT1	User programmable output (X)	63	OUT3	User programmable output (Z)
14	n.c.	No connection.	64	n.c.	No connection
15	n.c.	No connection	65	n.c.	No connection
16	AXIN1	User programmable input (X)	66	AXIN3	User programmable input (Z)
17	DACV1	Motor command (X), ±10V signal	67	DACV3	Motor command (Z), ±10V signal
18	AGND	Ground for motor command	68	AGND	Ground for motor command
19	A2+	Channel A+ encoder signal (Y)	69	A4+	Channel A+ encoder signal (W)
20	A2-	Channel A- inverted encoder signal (Y)	70	A4-	Channel A- inverted encoder signal (W)
21	B2+	Channel B+ encoder signal (Y)	71	B4+	Channel B+ encoder signal (W)
22	B2-	Channel B- inverted encoder signal (Y)	72	B4-	Channel B- inverted encoder signal (W)
23	I2+	Channel I+ encoder signal (Y)	73	I4+	Channel I+ encoder signal (W)
24	I2-	Channel I- inverted encoder signal (Y)	74	I4-	Channel I- inverted encoder signal (W)
25	Vc1	+5V	75	Vc1	+5V
26	GND	Ground	76	GND	Ground
27	XPLIM2	Limit switch positive direction (Y)	77	XPLIM4	Limit switch positive direction (W)
28	XNLIM2	Limit switch negative direction (Y)	78	XNLIM4	Limit switch negative direction (W)
29	HOME2	Home signal (Y) or high speed latch	79	HOME4	Home signal (W) or high speed latch
30	OUT2	User programmable output (Y)	80	OUT4	User programmable output (W)
31	n.c.	No connection	81	n.c.	No connection
32	n.c.	No connection	82	n.c.	No connection
33	AXIN2	User programmable input (Y)	83	AXIN4	User programmable input (W)
34	DACV2	Motor command (Y), ±10V signal	84	DACV4	Motor command (W), ±10V signal
35	AGND	Ground for motor command	85	AGND	Ground for motor command
36	UI0	uncommitted input (0)	86	UI4	uncommitted input (4)
37	UI1	uncommitted input (1)	87	UI5	uncommitted input (5)
38	UI2	uncommitted input (2)	88	UI6	uncommitted input (6)
39	UI3	uncommitted input (3)	89	UI7	uncommitted input (7)
40	AMPEN1	Amplifier enable (X) or user programmable output	90	AMPEN3	Amplifier enable (Z) or user programmable output
41	UO0	uncommitted output (0)	91	UO4	uncommitted output (4)

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
42	UO1	uncommitted output (1)	92	UO5	uncommitted output (5)
43	UO2	uncommitted output (2)	93	UO6	uncommitted output (6)
44	UO3	uncommitted output (3)	94	UO7	uncommitted output (7)
45	AMPEN2	Amplifier enable (Y) or user programmable output	95	AMPEN4	Amplifier enable (W) or user programmable output
46	RS-	Hardware reset	96	ANGND	GND for analog inputs
47	Ainput1	analog input channel (0)	97	Ainput5	analog input channel (4)
48	Ainput2	analog input channel (1)	98	Ainput6	analog input channel (5)
49	Ainput3	analog input channel (2)	99	Ainput7	analog input channel (6)
50	Ainput4	analog input channel (3)	100	Ainput8	analog input channel (7)

Pinout Beschreibung bürstenlose Servomotoren mit interne Kommutierung, Mikroschritt und Multiple Motor Modus; Option Connector POSYS® 192x

POSYS® 192x (68-position SCSI); Pinout-Beschreibung gilt nur in Verbindung mit internem und externem SCSI-Kabel auf IO68. Falls eigene Kabel und Boards verwendet werden, bitte Beschreibung unter Option Con verwenden.

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
1	BPWM1A	PWM magnitude Phase A (X)	2	BPWM1B	PWM magnitude Phase B (X)
3	BPWM1C	PWM magnitude Phase C (X)	4	GND	Ground
5	BPWMS1B	PWM sign (X)	6	GND	Ground
7	BPWM2A	PWM magnitude Phase A (Y)	8	BPWM2B	PWM magnitude Phase B (Y)
9	BPWM2C	PWM magnitude Phase C (Y)	10	GND	Ground
11	BPWMS2B	PWM sign (Y)	12	GND	Ground
13	BPWM3A	PWM magnitude Phase A (Z)	14	BPWM3B	PWM magnitude Phase B (Z)
15	BPWM3C	PWM magnitude Phase C (Z)	16	GND	Ground
17	BPWMS3B	PWM sign (Z)	18	GND	Ground
19	BPWM4A	PWM magnitude Phase A (W)	20	BPWM4B	PWM magnitude Phase B (W)
21	BPWM4C	PWM magnitude Phase C (W)	22	GND	Ground
23	BPWMS4B	PWM sign (W)	24	GND	Ground
25	HALL1A	Phase A Hall Sensor input (X)	26	HALL1B	Phase B Hall Sensor input (X)
27	HALL1C	Phase C Hall Sensor input (X)	28	GND	Ground
29	HALL2A	Phase A Hall Sensor input (Y)	30	HALL2B	Phase B Hall Sensor input (Y)
31	HALL2C	Phase C Hall Sensor input (Y)	32	GND	Ground
33	HALL3A	Phase A Hall Sensor input (Z)	34	HALL3B	Phase B Hall Sensor input (Z)
35	HALL3C	Phase C Hall Sensor input (Z)	36	GND	Ground
37	HALL4A	Phase A Hall Sensor input (W)	38	HALL4B	Phase B Hall Sensor input (W)
39	HALL4C	Phase C Hall Sensor input (W)	40	GND	Ground
41	Pulse1+	Pulse+ (X)	42	Pulse1-	Pulse- (X)
43	Pulse2+	Pulse+ (Y)	44	Pulse2-	Pulse- (Y)
45	Pulse3+	Pulse+ (Z)	46	Pulse3-	Pulse- (Z)
47	Pulse4+	Pulse+ (W)	48	Pulse4-	Pulse- (W)
49	Direction1+	Direction+ (X)	50	Direction1-	Direction- (X)
51	Direction2+	Direction+ (Y)	52	Direction2-	Direction- (Y)
53	Direction3+	Direction+ (Z)	54	Direction3-	Direction- (Z)
55	Direction4+	Direction+ (W)	56	Direction4-	Direction+ (W)
57	Vc1	+5V	58	Vc1	+5V
59	GND	Ground	60	GND	Ground
61	DAC1B	Motor command (X), ±10V signal, Phase B	62	AGND	Ground for motor command

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
63	DAC2B	Motor command (Y), ±10V signal, Phase B	64	AGND	Ground for motor command
65	DAC3B	Motor command (Z), ±10V signal, Phase B	66	AGND	Ground for motor command
67	DAC4B	Motor command (W), ±10V signal, Phase B	68	AGND	Ground for motor command

¤

Schrittmotor Pinout Beschreibung

Pinout Beschreibung Schritt-/Mikroschrittmotor* Version

POSYS® 195x

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
1	A1+	Channel A+ encoder signal (X)	51	A3+	Channel A+ encoder signal (Z)
2	A1-	Channel A- inverted encoder signal (X)	52	A3-	Channel A- inverted encoder signal (Z)
3	B1+	Channel B+ encoder signal (X)	53	B3+	Channel B+ encoder signal (Z)
4	B1-	Channel B- inverted encoder signal (X)	54	B3-	Channel B- inverted encoder signal (Z)
5	I1+	Channel I+ encoder signal (X)	55	I3+	Channel I+ encoder signal (Z)
6	I1-	Channel I- inverted encoder signal (X)	56	I3-	Channel I- inverted encoder signal (Z)
7	Vc1	+5V	57	Vc1	+5V
8	GND	Ground	58	GND	Ground
9	XPLIM1	Limit switch positive direction (X)	59	XPLIM3	Limit switch positive direction (Z)
10	XNLIM1	Limit switch negative direction (X)	60	XNLIM3	Limit switch negative direction (Z)
11	HOME1	Home signal (X) or high speed latch	61	HOME3	Home signal (Z) or high speed latch
12	GND	Ground	62	GND	Ground
13	AXISOUT1	User programmable output (X)	63	AXISOUT3	User programmable output (Z)
14	PULSE1	Pulse (X)	64	PULSE3	Pulse (Z)
15	DIR1	Direction (X)	65	DIR3	Direction (Z)
16	AXIN1	User programmable input (X)	66	AXIN3	User programmable input (Z)
17	ATREST1	At Rest (X)	67	ATREST3	At Rest (Z)
18	GND	Ground	68	GND	Ground
19	A2+	Channel A+ encoder signal (Y)	69	A4+	Channel A+ encoder signal (W)
20	A2-	Channel A- inverted encoder signal (Y)	70	A4-	Channel A- inverted encoder signal (W)
21	B2+	Channel B+ encoder signal (Y)	71	B4+	Channel B+ encoder signal (W)
22	B2-	Channel B- inverted encoder signal (Y)	72	B4-	Channel B- inverted encoder signal (W)
23	I2+	Channel I+ encoder signal (Y)	73	I4+	Channel I+ encoder signal (W)
24	I2-	Channel I- inverted encoder signal (Y)	74	I4-	Channel I- inverted encoder signal (W)
25	Vc1	+5V	75	Vc1	+5V
26	GND	Ground	76	GND	Ground
27	XPLIM2	Limit switch positive direction (Y)	77	XPLIM4	Limit switch positive direction (W)
28	XNLIM2	Limit switch negative direction (Y)	78	XNLIM4	Limit switch negative direction (W)

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
29	HOME2	Home signal (Y) or high speed latch	79	HOME4	Home signal (W) or high speed latch
30	AXISOUT2	User programmable output (Y)	80	AXISOUT4	User programmable output (W)
31	PULSE2	Pulse (Y)	81	PULSE4	Pulse (W)
32	DIR2	Direction (Y)	82	DIR4	Direction (W)
33	AXIN2	User programmable input (Y)	83	AXIN4	User programmable input (W)
34	ATREST2	At Rest (Y)	84	ATREST4	At Rest (W)
35	GND	Ground	85	GND	Ground
36	UI0	uncommitted input (0)	86	UI4	uncommitted input (4)
37	UI1	uncommitted input (1)	87	UI5	uncommitted input (5)
38	UI2	uncommitted input (2)	88	UI6	uncommitted input (6)
39	UI3	uncommitted input (3)	89	UI7	uncommitted input (7)
40	AMPEN1	Amplifier enable (X)	90	AMPEN3	Amplifier enable (Z)
41	UO0	uncommitted output (0)	91	UO4	uncommitted output (4)
42	UO1	uncommitted output (1)	92	UO5	uncommitted output (5)
43	UO2	uncommitted output (2)	93	UO6	uncommitted output (6)
44	UO3	uncommitted output (3)	94	UO7	uncommitted output (7)
45	AMPEN2	Amplifier enable (Y)	95	AMPEN4	Amplifier enable (W)
46	RS-	Hardware reset	96	ANGND	GND for analog inputs
47	Ainput1	analog input channel (0)	97	Ainput5	analog input channel (4)
48	Ainput2	analog input channel (1)	98	Ainput6	analog input channel (5)
49	Ainput3	analog input channel (2)	99	Ainput7	analog input channel (6)
50	Ainput4	analog input channel (3)	100	Ainput8	analog input channel (7)

*Mikroschritt mit geeignetem Treiber

Pinout Beschreibung Schrittmotor Version; Option Connector POSYS® 195x

POSYS® 195x (68-position SCSI)

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
1	n.c.	No Connection	2	n.c.	No Connection
3	n.c.	No Connection	4	GND	Ground
5	n.c.	No Connection	6	GND	Ground
7	n.c.	No Connection	8	n.c.	No Connection
9	n.c.	No Connection	10	GND	Ground
11	n.c.	No Connection	12	GND	Ground
13	n.c.	No Connection	14	n.c.	No Connection
15	n.c.	No Connection	16	GND	Ground
17	n.c.	No Connection	18	GND	Ground
19	n.c.	No Connection	20	n.c.	No Connection
21	n.c.	No Connection	22	GND	Ground
23	n.c.	No Connection	24	GND	Ground
25	n.c.	No Connection	26	n.c.	No Connection
27	n.c.	No Connection	28	GND	Ground
29	n.c.	No Connection	30	n.c.	No Connection
31	n.c.	No Connection	32	GND	Ground
33	n.c.	No Connection	34	n.c.	No Connection
35	n.c.	No Connection	36	GND	Ground
37	n.c.	No Connection	38	n.c.	No Connection
39	n.c.	No Connection	40	GND	Ground
41	Pulse1+	Pulse+ (X)	42	Pulse1-	Pulse- (X)

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
43	Pulse2+	Pulse+ (Y)	44	Pulse2-	Pulse- (Y)
45	Pulse3+	Pulse+ (Z)	46	Pulse3-	Pulse- (Z)
47	Pulse4+	Pulse+ (W)	48	Pulse4-	Pulse- (W)
49	Direction1+	Direction+ (X)	50	Direction1-	Direction- (X)
51	Direction2+	Direction+ (Y)	52	Direction2-	Direction- (Y)
53	Direction3+	Direction+ (Z)	54	Direction3-	Direction- (Z)
55	Direction4+	Direction+ (W)	56	Direction4-	Direction+ (W)
57	Vc1	+5V	58	Vc1	+5V
59	GND	Ground	60	GND	Ground
61	n.c.	No Connection	62	AGND	No Connection
63	n.c.	No Connection	64	AGND	No Connection
65	n.c.	No Connection	66	AGND	No Connection
67	n.c.	No Connection	68	AGND	No Connection

¤

Analog Eingang Pinout Beschreibung

Analog Eingang Pinout Beschreibung (nur gültig für POSYS® 1900)

J7 Pin-Out	Beschreibung	Erklärung
47	Analog Input 1	Analog input für channel 1
48	Analog Input 2	Analog input für channel 2
49	Analog Input 3	Analog input für channel 3
50	Analog Input 4	Analog input für channel 4
97	Analog Input 5	Analog input für channel 5
98	Analog Input 6	Analog input für channel 6
99	Analog Input 7	Analog input für channel 7
100	Analog Input 8	Analog input für channel 8

Analog1-8 sind 8 frei-programmierbare analoge Eingangssignale. Falls angeschlossen stellen die Spannungen an den Eingängen zwar noch nichts dar und der Motion Prozessor reagiert nicht darauf, allerdings stellen sie eine gute Möglichkeit dar, dass des Anwender's Programm interaktiv hierrauf reagiert. Diese Signale werden mit dem Kommando ReadAnalog gelesen.

Die minimal erlaubte Eingangsspannung beträgt 0,0V und die maximale erlaubte Eingangsspannung beträgt 2.048V. Die Auflösung beträgt 10 Bit. Um den numerischen Wert zu bestimmen kann folgende Formel verwendet werden:

ReadValue(Channel) = (AnalogVoltage * 65,536) / 2.048V

Umgekehrt wird mit folgender Formel die entsprechende Spannung berechnet:

AnalogVoltage = (ReadValue(Channel) * 2.048V) / 65,536

Damit die Analogspannungen korrekt ermittelt werden können, müssen die Analogsignale zusätzlich mit AnalogGND (Analog Ground) verbunden sein.

Falls andere als die standardmäßigen Eingangsspannungen gewünscht sind, können nach Absprache folgende Spannungen ersatzweise und optional angeboten werden: 1.8V, 2.5V, 3.0V und 3.3V.

¤

GP CON Steckverbinder

GP CON Steckverbinder

„GP CON“ Stecker

Dieser Stecker wird verwendet um verschiedene Peripherie Komponenten, wie z.B. Enkoder, Verstärker, IOs etc. anzuschließen. Es ist ein 100-poliger High-Density Stecker. Die folgende Tabelle bietet Bestellinformationen, falls Sie ihr eigenes Kabel konfektionieren möchten (POSYS® 7xx/9xx/19xx):

Artikelbeschreibung	Hersteller und Artikelnummer
Stecker mit geraden Pins	AMP: 2-5175887-9 (Champ)
	3M: 102A0-6212VC (Mini D-ribbon)
Stecker mit rechtwinkligen Pins	AMP: 2-5178238-9 (Champ)
	3M: N102A0-52E2VC
Geschirmte Kabel Stecker	AMP: 176793-9 champ (Gehäuse) AMP: 2-5175677-9 champ (Stecker)
	3M: 103A0-A200-00 (Gehäuse) 3M: 101A0-6000EC (Stecker)

Stecker auf den POSYS® 800-B/1800 Karten (PC/104):
TSM-125-01-T-DV (Samtec)
Mating with: IDSD-25-S

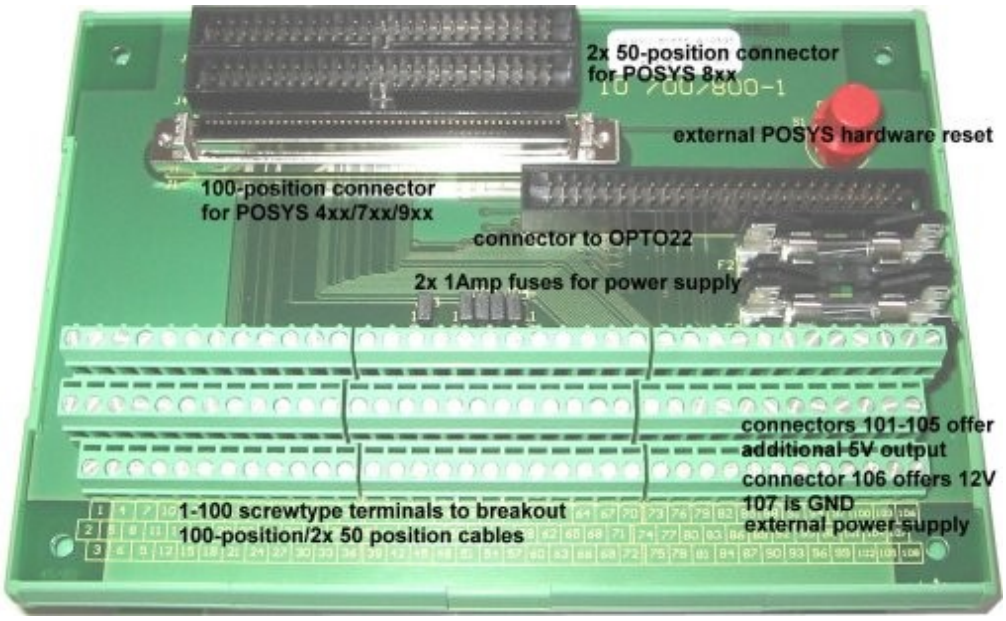
Je nachdem welche Karte eingesetzt wird, gibt es unterschiedliche Pinbelegungen. Diese sind in den Kapiteln POSYS® 7xx, POSYS® 8xx-B, POSYS® 18xx, POSYS® 9xx und POSYS® 19xx beschrieben.

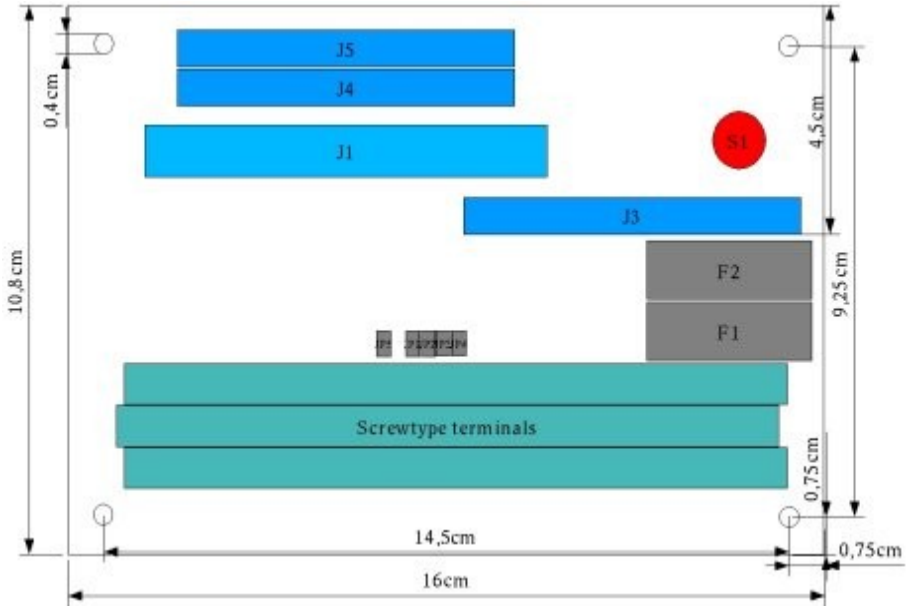
¤

Interconnect Module

IO700/800-1

IO700/800-1





Der IO700/800-1 ist ein Interconnectmodul zum leichteren Anschliessen der Peripheriekomponenten, wie Verstärker, Enkoder, E/As usw. Es stellt den zentralen Punkt zum Anschliessen dieser Komponenten dar.

Das Gehäuse der Platine ist ein Standard Phoenix System für die Hutschienenmontage. Als Alternative stellt die Platine 4 Löcher zur Befestigung an eine beliebige Unterlage zur Verfügung. Die Löcher haben ein Durchmesser von 3,5 mm.

Die Pinbelegung der POSYS® auf den vorhergehenden Seiten sind identisch mit den Belegungen auf der Platine für die Schraubklemmenanschlüsse. Dies erleichtert zusätzlich die Orientierung. Bitte achten Sie auf die Unterschiede je nach Motortyp.

J1 wird mit der ISA-Bus POSYS® 7xx und den PCI-Bus Positionierteuerungen POSYS 9xx/19xx verwendet.

J4 und J5 sind parallel zu J1 angeordnet und werden mit den PC/104 Positioniersteuerungen POSYS® 8xx/8xx-B und 18xx verwendet. J5 entspricht den Pinouts 1-50 und J4 entspricht den Pinouts 51-100. Bitte ziehen Sie die Pinout Beschreibung für die einzelnen PC/104 Positioniersteuerungen für korrekte Verkabelung zu Rate.

Die POSYS® verwendet die 5V des Computers und stellt sie an den Pins 7, 25, 57 und 75 für die Enkoderversorgung zur Verfügung. Der Ausgang ist mit einer 1 Ampere Sicherung gesichert. Es ist auch möglich eine externe Spannungsversorgung zu verwenden. Die Steckbrücken JP1, JP2, JP3 and JP4 sind stardardmäßig auf den Pins 1 - 2 gesteckt wenn die interne Spannungsversdogung über die Karte verwendet werden soll. Die Anschlüsse 101, 102, 103, 104 und 105 stellen nochmal eine Anschlussmöglichkeit dar um eine externe Spannungsversogung von entweder 5V oder 12V für Enkoder zur Verfügung zu stellen. In diesem Fall müssen die Steckbrücken JP1 - JP4 auf Pins 2 - 3 gesteckt werden. Auch dieser Eingang ist mit einer 1 Ampere Sicherung gesichert. Je nachdem welcher Enkodertyp verwendet wird, können daher entweder 5V oder max. 12V für Enkoder die mehr als 5V benötigen, verwendet werden. In diesem Fall muß die externe Spannung an Pin 106 und die Erdung an Pin 107 angeschlossen werden.

Die mechanischen Dimensionen des Moduls sind wie folgt:

- mit Phoenix Gehäuse: 162x112x50 mm
- ohne Phoenix Gehäuse (nur Platine): 160x108x28 mm

OPTO 22

Die Verbindungen zu J3 (OPTO 22) sind wie folgt:

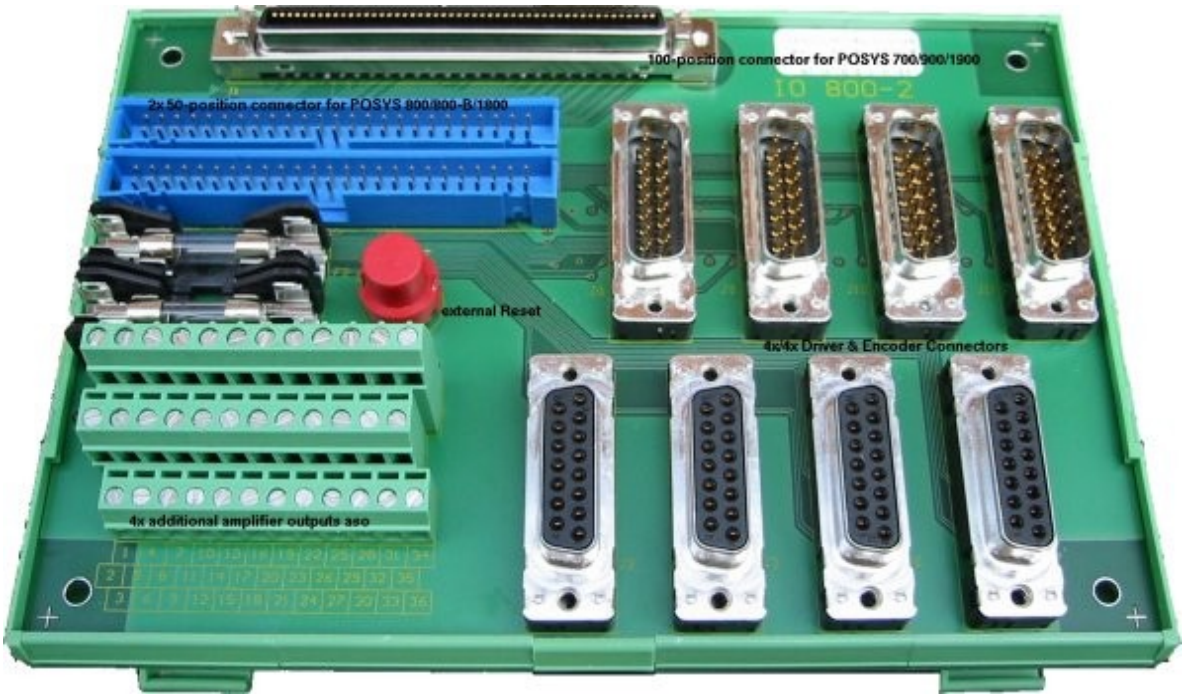
J3, Pins 2, 4, 6, 8…50 sind mit GND verbunden
Steckbrücke JP5, Pin 49 von J3 wird mit 5 VDC versorgt, falls die Steckbrücke gesteckt ist.

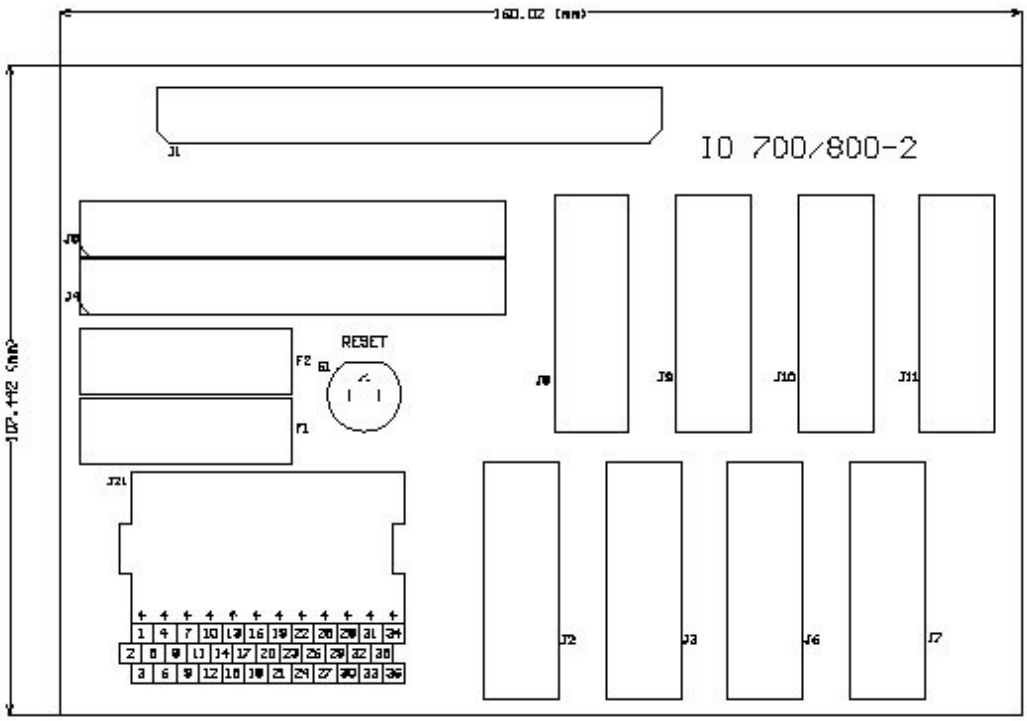
	J1	J3	Beschreibung
Pin		49	Verbunden mit JP5, stellt +5 VDC, falls gesteckt, zur Verfügung
	36	47	UI0
	J1	J3	Beschreibung
Pin	37	45	UI1
	38	43	UI2
	39	41	UI3
	86	39	UI4
	87	37	UI5
	88	35	UI6
	89	33	UI7
	41	31	UO0
	42	29	UO1
	43	27	UO2
	44	25	UO3
	91	23	UO4
	92	21	UO5
	93	19	UO6
	94	17	UO7
	40	15	AMPEN1
	45	13	AMPEN2
	90	11	AMPEN3
	95	09	AMPEN4
	13	07	OUT1
	30	05	OUT2
	63	03	OUT3
	80	01	OUT4

¤

IO700/800-2

Dieses Interconnectmodul wurde speziell in Zusammenarbeit mit Berger Lahr entwickelt um eine Schnittstelle zu den TLCC Schrittmotor (Takt & Richtung) und Servomotor (±10V Analogsignalausgang) Modulen der Firma Berger Lahr zur Verfügung zu stellen. Bei entsprechender Kabelkonfektionierung ist auch die Verwendung dieses Moduls mit Verstärkern und Treibern anderer Hersteller möglich. Hierzu verweisen wir auf die Beschreibung der Hersteller der Verstärker und Module die evtl. zum Einsatz kommen sollen. Die folgende Tabelle listet die Belegungen für die einzelnen Achsen auf:

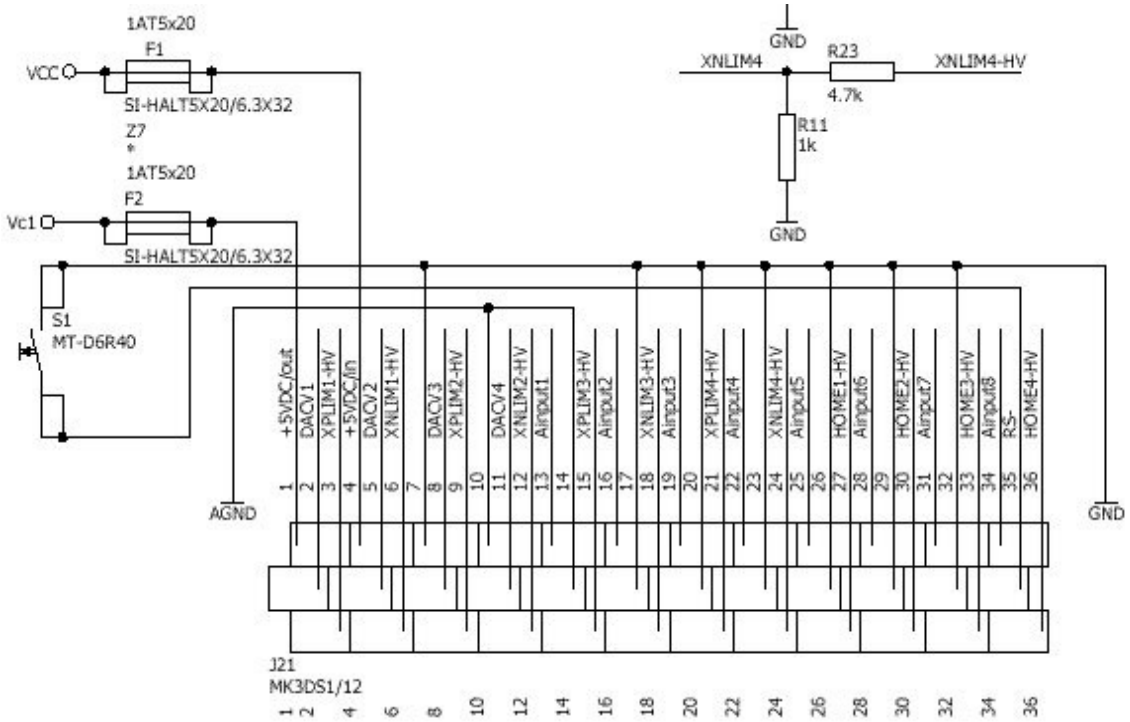

100-position connector for POSYS 700/900/1900
IO 800-2
2x50-position connector for POSYS 800/800-B/1800
external Reset
4x4x Driver & Encoder Connectors
4x additional amplifier outputs aso



Die folgende Tabelle listet die Beschreibung und Belegung im Einzelnen auf:

Verbindungen	Beschreibung				
J1	100-poliger Stecker für Standard POSYS®-Kabel zur POSYS® 700/900/1900 und mit Adapter auch zu POSYS® 800-B/1800. Kabel ist rund und geschirmt und hat Klemmanschlüsse für den sicheren Halt				
J4, J5	2 x 50-poliger IDC Stecker-Buchse. Stellt die gleichen Verbindungen zur Verfügung wie J1 und ist für POSYS® 800-B/1800 gedacht ohne Adapterkarte. J4 ist für die Achsen 1+2 (X/Y) und J5 ist für die Ach 3+4 (Z/W). Die Anschlüsse von J4 (Pins 1-50) sind parallel zu den Anschlüssen 1-50 von J1. Die Anschlüsse von J5 (Pins 1-50) sind parallel zu den Anschlüssen 51-100 von J1.				
J2, J3,J6, J7	4 x 15-poliger SUB-D Buchse für Enkoder.				
	PIN	J2	J3	J6	J7
	1	A1+	A2+	A3+	A4+
	9	A1-	A2-	A3-	A4-
	12	B1+	B2+	B3+	B4+
	5	B1-	B2-	B3-	B4-
	13	I1+	I2+	I3+	I4+
	6	I1-	I2-	I3-	I4-
	2	VCC	VCC	VCC	VCC
	3	GND	GND	GND	GND
	Die Spannungsversorgung für die Enkoder kann entweder 5 or 12 VDC , abhängig vom Typ der verwendeten Enkoder.				
J8 – J11	4 x 15-poliger SUB-D Stecker für Takt & Richtungssignale, Verstärkerfreigabe, AxisOUT Ausgang, Axis Eingang und UI-Eingang				
	Pin	J8	J9	J10	J11
	1	Pulse1+	Pulse2+	Pulse3+	Pulse4+
	9	Pulse1-	Pulse2-	Pulse3-	Pulse4-
	2	DIR1+	DIR2+	DIR3+	DIR4+
	10	DIR1-	DIR2-	DIR3-	DIR4-
	3	AMPEN1+	AMPEN2+	AMPEN3+	AMPEN4+
	11	AMPEN1-	AMPEN2-	AMPEN3-	AMPEN4-
	5	OUT1+	OUT2+	OUT3+	OUT4+
	6	OUT1-	OUT2-	OUT3-	OUT4-
	8	AxisIn1	AxisIn2	AxisIn3	AxisIn4
	15	GND	GND	GND	GND
J21	J21 ist ein Schraubklemmenanschluss mit 36 Anschlüssen. Die Anschlüsse sind wie folgt belegt:				

Verbindungen	Beschreibung



Pin	Name	Besschreibung
1	5VDC/out	5 V Ausgang vom Computer. Kann mit Anschluss 4 verbunden werden um Enkoder mit 5V zu versorge
2	DACV1	±10V Motor Ausgangssignal (nur für Servomotoren X-Achse)
3	XPLIM1-HV	positiver Endschaltereingang, X-Achse, 24VDC ma
4	5VDC/in	VCC Eingang für Enkoderspannungs- versorgung (VDC or 12 VDC)
5	DACV2	±10V Motor Ausgangssignal (nur für Servomotoren Y-Achse)
6	XNLIM1-HV	negativer Endschaltereingang, X-Achse, 24VDC ma
7	GND	Digital Ground
8	DACV3	±10V Motor Ausgangssignal (nur für Servomotoren Z-Achse)
9	XPLIM2-HV	positiver Endschaltereingang, Y-Achse, 24VDC ma
10	AGND	Analog Ground für analoge Ausgänge und Eingänge
11	DACV4	±10V Motor Ausgangssignal (nur für Servomotoren W-Achse)
12	XNLIM2-HV	negativer Endschaltereingang, Y-Achse, 24VDC ma
13	AINPUT1	Analoger Eingangskanal 1 (0-2.5 VDC)
14	AGND	Analog Ground für analoge Ausgänge und Eingänge
15	XPLIM3-HV	positiver Endschaltereingang, Z-Achse, 24VDC ma
16	AINPUT2	Analoger Eingangskanal 2 (0-2.5 VDC)
17	GND	Digital Ground
18	XNLIM3-HV	negativer Endschaltereingang, Z-Achse, 24VDC ma
19	AINPUT3	Analoger Eingangskanal 3 (0-2.5 VDC)
20	GND	Digital Ground
21	XPLIM4-HV	positiver Endschaltereingang, W-Achse, 24VDC ma
22	AINPUT4	Analoger Eingangskanal 4 (0-2.5 VDC)
23	GND	Digital Ground
24	XNLIM4-HV	negativer Endschaltereingang, W-Achse, 24VDC max.

Verbindungen	Beschreibung	
25	AINPUT5	Analoger Eingangskanal 5 (0-2.5 VDC)
26	GND	Digital Ground
27	HOME1-HV	Latch Input (High Speed), X-Achse, 24 VDC max.
28	AINPUT6	Analoger Eingangskanal 6 (0-2.5 VDC)
29	GND	Digital Ground
30	HOME2-HV	Latch Input (High Speed), Y-Achse, 24 VDC max.
31	AINPUT7	Analoger Eingangskanal 7 (0-2.5 VDC)
32	GND	Digital Ground
33	HOME3-HV	Latch Input (High Speed), Z-Achse, 24 VDC max.
34	AINPUT8	Analoger Eingangskanal 8 (0-2.5 VDC)
35	RS-	Reset input, parallel to the Reset switch S1
36	HOME4-HV	Latch Input (High Speed), W-Achse, 24 VDC max.

Die analogen Ausgänge DAC V1 bis DAC V4 werden nur auf C ontrollern für Servomotoren die ±10 V Drehmomentsollv
benötigen, zur Verfügung gestellt. Motion C ontroller für Schrittmotoren stellen nur Takt & Richtungssignale zur Verfüg
Die neue POSYS® Serie 180x/190x/182x/192x erlaubt die Verwendung beider Signaltypen.
F 1 und F 2 sind 1 Ampere Sicherungen. Die 5 VDC vom Host und die externen 5VDC und 12VDC werden gesichert.

¤

OPTION CON

Falls DC bürstenlose Servomotoren oder Mikroschrittmotoren verwendet werden, wird auch der „Option Con“ Stecker für die zusätzlichen Signale (multiphasen Motorsignalausgabe und Eingänge für z.B. Hallsensoren) benötigt. Sobald nur eine Achse für DC bürstenlose Servomotoren, Schrittmotoren mit invertierter Signalausgabe oder Mikroschrittmotoren verwendet wird, benötigen Sie dieses zusätzliche Kabel (bei Verwendung der On-Board Kommutierung oder Sinus/Cosinus Signalausgabe für Mikroschrittmotoren).

Der „Option Con“ Stecker ist ein 68-poliger SCSI Stecker für die PCI-Karten und ein 50-poliger IDC Stecker für PC/104 Karten (POSYS® 18xx). Die folgende Tabelle listet die notwendigen Informationen, für jene Kunden auf, die ihr eigenes Kabel für PCI-Karten konfektionieren möchten:

Teilebeschreibung	Hersteller und Artikelnummer
Flachbandkabel	Harting: 60 06 068 5440 (für AWG 30, 0,635mm Flachbandkabel)
Stecker	Harting: 60 03 068 52
Gehäuse	Harting: 60 03 068 0155

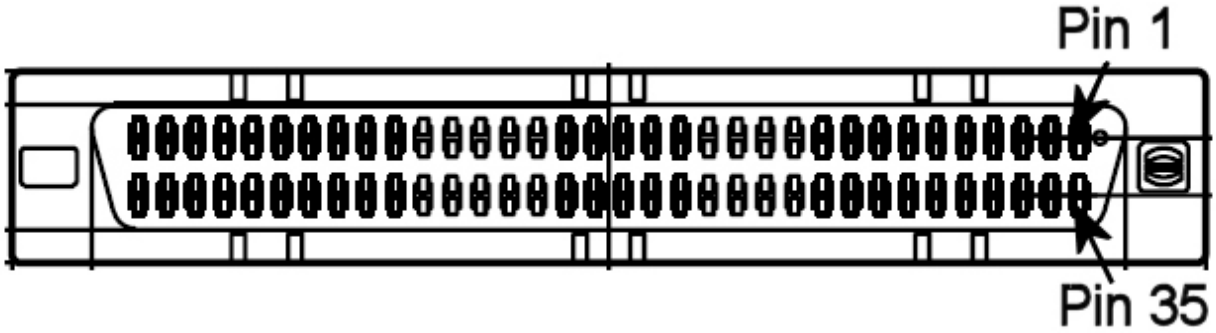


Bild zeigt OPTION CON (nur PCI-Karten) mit Pinbelegung (Draufsicht)

Die folgenden Tabellen zeigen die Belegung des „Option Con“ bei Verwendung von DC bürstenlosen Servo-, Schrittmotoren mit invertierter Signalausgabe bzw. Mikroschrittmotoren. Wenn nicht anders beschrieben steht die Zahl „1“ für Achse1, „2“ für Achse 2 usw., und A steht für Phase A, B für Phase B usw.

POSYS® PCI-Karten

Beschreibung	Pinnummer	Pinnummer	Beschreibung
PWM Magnitude 1A	1	35	PWM Magnitude 1B
PWM Magnitude 1C	2	36	GND
PWM Sign 1	3	37	GND
PWM Magnitude 2A	4	38	PWM Magnitude 2B
PWM Magnitude 2C	5	39	GND
PWM Sign 2	6	40	GND
PWM Magnitude 3A	7	41	PWM Magnitude 3B
PWM Magnitude 3C	8	42	GND
PWM Sign 3	9	43	GND
PWM Magnitude 4A	10	44	PWM Magnitude 4B
PWM Magnitude 4C	11	45	GND
PWM Sign 4	12	46	GND
Hall 1A	13	47	Hall 1B
Hall 1C	14	48	GND
Hall 2A	15	49	Hall 2B
Hall 2C	16	50	GND
Hall 3A	17	51	Hall 3B

Beschreibung	Pinnummer	Pinnummer	Beschreibung
Hall 3C	18	52	GND
Hall 4A	19	53	Hall 4B
Hall 4C	20	54	GND
Pulse 1+	21	55	Pulse 1-
Pulse 2+	22	56	Pulse 2-
Pulse 3+	23	57	Pulse 3-
Pulse 4+	24	58	Pulse 4-
Direction 1+	25	59	Direction 1-
Direction 2+	26	60	Direction 2-
Direction 3+	27	61	Direction 3-
Direction 4+	28	62	Direction 4-
VCC (5V) Output	29	63	VCC (5V) Output
GND	30	64	GND
DACB1 (X-Axis)	31	65	DACA1 (X-Axis)
DACB2 (Y-Axis)	32	66	DACA2 (Y-Axis)
DACB3 (Z-Axis)	33	67	DACA3 (Z-Axis)
DACB4 (W-Axis)	34	68	DACA4 (W-Axis)

POSYS® PC/104 Karten

Die folgende Tabelle listet die notwendigen Informationen, für jene Kunden auf, die ihr eigenes Kabel für PC/104-Karten konfektionieren möchten:

Artikelbeschreibung	Hersteller und Artikelnummer
POSYS® PC/104: Buchse IDC mit Kabel 50-polig	Samtec: ASP-53116-02

Der Option CON ist ein 50-poliger IDC-Stecker für die POSYS® 182x/185x:

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
1	BPWM1A	PWM magnitude Phase A (X)	26	HALL2B	Phase B Hall Sensor input (Y)
2	BPWM1B	PWM magnitude Phase B (X)	27	HALL2C	Phase C Hall Sensor input (Y)
3	BPWM1C	PWM magnitude Phase C (X)	28	GND	Ground
4	BPWMS1A	PWM sign Phase A (X)	29	HALL3A	Phase A Hall Sensor input (Z)
5	GND	Ground	30	HALL3B	Phase B Hall Sensor input (Z)
6	BPWM2A	PWM magnitude Phase A (Y)	31	HALL3C	Phase C Hall Sensor input (Z)
7	BPWM2B	PWM magnitude Phase B (Y)	32	GND	Ground
8	BPWM2C	PWM magnitude Phase C (Y)	33	HALL4A	Phase A Hall Sensor input (W)
9	BPWMS2A	PWM sign Phase A (Y)	34	HALL4B	Phase B Hall Sensor input (W)
10	GND	Ground	35	HALL4C	Phase C Hall Sensor input (W)
11	BPWM3A	PWM magnitude Phase A (Z)	36	GND	Ground
12	BPWM3B	PWM magnitude Phase B (Z)	37	AGND	Ground for motor command
13	BPWM3C	PWM magnitude Phase C (Z)	38	DAC1A	Motor command (X), ±10V signal, Phase A
14	BPWMS3A	PWM sign Phase A (Z)	39	DAC2A	Motor command (Y), ±10V signal, Phase A
15	GND	Ground	40	DAC3A	Motor command (Z), ±10V signal, Phase A
16	BPWM4A	PWM magnitude Phase A (W)	41	DAC4A	Motor command (W), ±10V signal, Phase A
17	BPWM4B	PWM magnitude Phase B (W)	42	DAC1B	Motor command (X), ±10V signal, Phase B
18	BPWM4C	PWM magnitude Phase C (W)	43	DAC2B	Motor command (Y), ±10V signal, Phase B
19	BPWMS4A	PWM sign Phase A (W)	44	DAC3B	Motor command (Z), ±10V signal, Phase B

Pin	Verbindung	Beschreibung	Pin	Verbindung	Beschreibung
20	GND	Ground	45	DAC4B	Motor command (W), ±10V signal, Phase B
21	HALL1A	Phase A Hall Sensor input (X)	46	AGND	Ground for motor command
22	HALL1B	Phase B Hall Sensor input (X)	47	GND	Ground
23	HALL1C	Phase C Hall Sensor input (X)	48	GND	Ground
24	GND	Ground	49	Vc1	+5V
25	HALL2A	Phase A Hall Sensor input (Y)	50	Vc1	+5V

¤

Verbindungsübersicht - Motorverstärker

Überblick

Anschlussübersicht – Motorverstärker

Die POSYS® Karte unterstützt 4 verschiedene Motorausgangssignalarten für Verstärker:

Motorausgangssignalart	Beschreibung
DAC	analoge Signale von den on-board D/A Konvertern
PWM Sign / Magnitude	Impuls-breiten-modulierte Signale differenziert nach Magnitude und Richtung pro Phase.
PWM 50/50	Impuls-breiten-moduliertes Signal mit einzelnem Rechtecksignal pro Phase
Pulse & Direction	Takt & Richtungssignal

Zu jeder Achse können, je nach Motortyp, 1, 2 oder 3 Phasen Verwendung finden. DC bürstenbehaftete Motoren haben 1 Phase, wobei bei DC bürstenlosen Servomotoren oder Mikroschrittmotoren 2 oder 3 Phasen, abhängig von der Signalart, verwendet werden.

Die folgenden Tabellen listen, abhängig von verwendeten Motoren und Motorausgangssignalart, die grundlegenden Verbindungsmöglichkeiten auf. Diese Ausgänge sollten von den bezeichneten Pins auf die entsprechenden Verstärkereingänge gelegt werden. Beachten Sie bitte, dass die Namen der Pins von den Namen die die Verstärkerhersteller verwenden, abweichen können. Es werden gebräuchliche Namen verwendet.

¤

POSYS® 7xx

POSYS® 7xx; Verbindungsübersicht - Motorverstärker

POSYS® 70x Serie, für bürstenbehaftete Servomotoren mit DAC-Ausgang, PWM-Ausgang (sign/magnitude oder 50/50) und bürstenlose Servomotoren mit externe Kommutierung mit DAC-Ausgang, PWM-Ausgang (sign/magnitude oder 50/50).

GP CON J6, 100-poliger Stecker

Motorausgangs-signal	Signal	Verbindung zu Verstärker	X-Achse	Y-Achse	Z-Achse	W-Achse
16 Bit DAC (±10 Volt)	DACV 1-4	Ref+ or V+	17	34	67	84
Analog Ground	AGND	Ref- or GND	18	35	68	85
PWM Sign + Magnitude oder 50/50 Output	BPWM 1-4	PWM Magnitude	14	31	64	81
PWM Sign	BPWMS 1-4	PWM Sign	15	32	65	82
Digital Ground für 50/50	GND	GND	12	26	62	76

POSYS® 75x Serie, Schrittmotormodus mit Takt & Richtungssignal; bei Verwendung geeigneter Treiber ist diese Karte auch als Mikroschrittcontroller verwendbar.

GP CON J6, 100-poliger Stecker

Motorausgangs-signal	Signal	Verbindung zu Verstärker	X-Achse	Y-Achse	Z-Achse	W-Achse
Pulse (TTL)	Pulse 1-4	Pulse or Clock	14	31	64	81
Direction (TTL)	Dir 1-4	Direction input	15	32	65	82
In–Position (TTL)	AtRest 1-4	Standby Current	17	34	67	84

¤

POSYS® 8xx-B/18xx

POSYS® 8xx-B/18xx; Verbindungsübersicht - Motorverstärker

Die POSYS® PC/104 Karten unterstützen je nach Modell bis zu 4 verschiedene Motorausgangssignalmethoden:

Motorausgangssignalart	Beschreibung
DAC	analoge Signale von den on-board D/A Konvertern
PWM Sign / Magnitude	Impuls-breiten-modulierte Signale differenziert nach Magnitude und Richtung pro Phase.
PWM 50/50	Impuls-breiten-moduliertes Signal mit einzelnem Rechtecksignal pro Phase
Takt & Richtung	Takt & Richtungssignal

Zu jeder Achse können, je nach Motortyp, 1, 2 oder 3 Phasen Verwendung finden. DC bürstenbehaftete Motoren haben 1 Phase, wobei bei DC bürstenlosen Servomotoren oder Mikroschrittmotoren 2 oder 3 Phasen, abhängig von der Signalform, verwendet werden.

Die folgenden Tabellen listen, abhängig von verwendeten Motoren und Motorausgangssignalart, die grundlegenden Verbindungsmöglichkeiten auf. Diese Ausgänge sollten von den bezeichneten Pins auf die entsprechenden Verstärkereingänge gelegt werden. Beachten Sie bitte, dass die Namen der Pins von den Namen die die Verstärkerhersteller verwenden, abweichen können. Es werden gebräuchliche Namen verwendet.

Bürstenbehafteter Servomotormodus mit ±10V Analogausgang oder PWM-Ausgang (Sign/Magnitude oder 50/50) (externe Kommutierung)

POSYS® 80x-B
POSYS® 182x (für korrekte Interpretation: J7 = J5 und J1 = J6)

			Connection			
Motor output method	POSYS® 80x-B/182x connection name	Amplifier connection	Axis 1	Axis 2	Axis 3	Axis 4
DAC	DAC 1-4A	Ref+ or V+	J7-17	J7-34	J1-17	J1-34
	AGND	Ref- or Gnd	J7-18	J7-35	J1-18	J1-35
PWM sign/magnitude	PWMMag1-4A	PWM magnitude	J7-14	J7-31	J1-14	J1-31
	PWMSign1-4A	PWM direction	J7-15	J7-32	J1-15	J1-32
	GND	Gnd	J7-8	J7-26	J1-8	J1-26
PWM 50/50	unused					

Bürstenloser Servomotormodus mit interne Kommutierung

POSYS® 182x (für korrekte Interpretation: J7 = J5 und J1 = J6)

			Connection			
Motor output method	POSYS® 182x connection name	Amplifier connection	Axis 1	Axis 2	Axis 3	Axis 4
DAC	DAC 1-4A	Ref1+ or V1+	Option Con-38	Option Con-39	Option Con-40	Option Con-41
	DAC1-4B	Ref2+ or V2+	Option Con-42	Option Con-43	Option Con-44	Option Con-45
	AGND	Ref- or Gnd	Option Con-37	Option Con-46	Option Con-37	Option Con-46
PWM 50/50	PWMMag1-4A	PWM phase 1	Option Con-1	Option Con-6	Option Con-11	Option Con-16J1-31
	PWMMag1-4B	PWM phase 2	Option	Option	Option	Option

			Connection			
Motor output method	POSYS® 182x connection name	Amplifier connection	Axis 1	Axis 2	Axis 3	Axis 4
			Con-2	Con-7	Con-12	con-17
	PWMMag1-4C	PWM phase 3	Option Con-3	Option Con-8	Option Con-13	Option Con-18
	GND	Gnd	Option Con-5	Option Con-10	Option Con-15	Option Con-20
PWM sign/magnitude	unused					

Mikroschrittmotormodus

POSYS® 182x (für korrekte Interpretation: J7 = J5 und J1 = J6)

			Connection			
Motor output method	POSYS® 18xx connection name	Amplifier connection	Axis 1	Axis 2	Axis 3	Axis 4
DAC	DAC 1-4A	Ref1+ or V1+	Option Con-38	Option Con-39	Option Con-40	Option Con-41
	DAC1-4B	Ref2+ or V2+	Option Con-42	Option Con-43	Option Con-44	Option Con-45
	AGND	Ref- or Gnd	Option Con-37	Option Con-46	Option Con-37	Option Con-46
PWM sign/magnitude	PWMMag1-4A	PWM magnitude	Option Con-1	Option Con-6	Option Con-11	Option Con-16
	PWMSign1-4A	PWM direction	Option Con-4	Option Con-9	Option Con-14	Option Con-19
	PWMMag1-4B	PWM magnitude	Option Con-2	Option Con-7	Option Con-12	Option Con-17
	PWMSign1-4B	PWM sign	Option Con-3	Option Con-8	Option Con-13	Option con-18
	GND	Gnd	Option Con-5	Option Con-10	Option Con-15	Option Con-20
PWM 50/50	unused					

Schrittmotormodus, TTL level Takt & Richtung

POSYS® 85x-B
POSYS® 182x/185x (für korrekte Interpretation: J7 = J5 und J1 = J6)

			Connection			
Motor output method	POSYS® 85x-B/ 18xx connection name	Amplifier connection	Axis 1	Axis 2	Axis 3	Axis 4
Pulse & Direction	Pulse 1-4	Pulse or step	J7-14	J7-31	J1-14	J1-31
	Direction 1-4	Direction	J7-15	J7-32	J1-15	J1-32
	GND	Gnd	J7-8	J7-26	J1-8	J1-26

¤

POSYS® 9xx/19xx

POSYS® 9xx/19xx; Verbindungsübersicht – Motorverstärker

Die POSYS® PCI-Karte unterstützt je nach Modell bis zu 4 verschiedene Motorausgangssignalarten für Verstärker:

Motorausgangssignalart	Beschreibung
DAC	analoge Signale von den on-board D/A Konvertern
PWM Sign / Magnitude	Impuls-breiten-modulierte Signale differenziert nach Magnitude und Richtung pro Phase.
PWM 50/50	Impuls-breiten-moduliertes Signal mit einzelnem Rechtecksignal pro Phase
Pulse & Direction	Takt & Richtungssignal

Zu jeder Achse können, je nach Motortyp, 1, 2 oder 3 Phasen Verwendung finden. DC bürstenbehaftete Motoren haben 1 Phase, wobei bei DC bürstenlosen Servomotoren oder Mikroschrittmotoren 2 oder 3 Phasen, abhängig von der Signalform, verwendet werden.

Die folgenden Tabellen listen, abhängig von verwendeten Motoren und Motorausgangssignalart, die grundlegenden Verbindungsmöglichkeiten auf. Diese Ausgänge sollten von den bezeichneten Pins auf die entsprechenden Verstärkereingänge gelegt werden. Beachten Sie bitte, dass die Namen der Pins von den Namen die die Verstärkerhersteller verwenden, abweichen können. Es werden gebräuchliche Namen verwendet.

Bürstenbehafteter Servomodus mit Analogausgang, PWM-Ausgang (Sign/Magnitude oder 50/50)

POSYS® 90x/91x/97x/98x
POSYS® 190x/192x

GP CON J6, 100-poliger Stecker

Motorausgabe-methode	Signal	Verbindung zum Verstärker	X-Achse	Y-Achse	Z-Achse	W-Achse
16 Bit DAC (±10 Volt)	DACV 1-4	Ref+ or V+	17	34	67	84
Analog Ground	AGND	Ref- or GND	18	35	68	85
PWM Sign + Magnitude oder 50/50 Output	BPWM 1-4	PWM Magnitude	14	31	64	81
PWM Sign	BPWMS 1-4	PWM Sign	15	32	65	82
Digital Ground für 50/50	GND	GND	12	26	62	76

Schrittmotormodus, TTL level Takt & Richtung

POSYS® 95x
POSYS® 190x/192x/195x

GP CON J6, 100-poliger Stecker

Motorausgabe-methode	Signal	Verbindung zum Verstärker	X-Achse	Y-Achse	Z-Achse	W-Achse
Pulse (TTL)	Pulse 1-4	Pulse or Clock	14	31	64	81
Direction (TTL)	Dir 1-4	Direction input	15	32	65	82
In –Position (TTL)	AtRest 1-4	Standby Current	17	34	67	84

Schrittmotormodus, invertierte Signale Takt & Richtung

POSYS® 95x
POSYS® 190x/192x/195x (Standard 192x/195x)

Diese Ausgänge sind nur verfügbar, wenn die Karte mit dem „Option Con“ J5 ausgestattet ist.

OPTION CON J5, 68-poliger Stecker

Motorausgabe-methode	Signal	Verbindung zum Verstärker	X-Achse	Y-Achse	Z-Achse	W-Achse
Pulse, (TTL)	Pulse 1-4	Pulse oder Clock	1	4	7	10
Direction, (TTL)	Dir 1-4	Direction input	3	6	9	12
Pulse+, Line Driver	Pulse+ 1-4	Pulse+ oder Clock+	21	22	23	24

Motorausgabe-methode	Signal	Verbindung zum Verstärker	X-Achse	Y-Achse	Z-Achse	W-Achse
Pulse- , Line Driver	Pulse – 1-4	Pulse– oder Clock-	55	56	57	58
Direction+, Line Driver	Dir+ 1-4	Direction+	25	26	27	28
Direction-, Line Driver	Dir- 1-4	Direction-	59	60	61	62
In-Position, TTL	AtRest 1-4	Standby Current	2	5	8	11

Bürstenloser Servomotormodus mit interner Kommutierung

POSYS® 92x/93x/97x/98x
POSYS® 192x

Diese Ausgänge sind nur verfügbar, wenn die Karte mit dem „Option Con“ J5 ausgestattet ist.

GP CON J6, 100-poliger Stecker

Motorausgabe-methode	Signal	Verbindung zum Verstärker	X-Achse	Y-Achse	Z-Achse	W-Achse
16 Bit DAC ± 10 Volt	DACV 1-4	Ref+ or V+	17	34	67	84
Analog Ground	AGND (1-4)	Ref- or GND	18	35	68	85

In diesem Modus sind DACV1-4 und AGND (1-4) doppelt vorhanden (GP CON und OPTION CON), aber nur das eine oder andere muss angeschlossen werden. Der Anwender kann entscheiden welche Anschlüsse für seine Applikation am bequemsten zu verwenden sind.

OPTION CON J5, 68-poliger Stecker

Motorausgabe-methode	Signal	Verbindung zum Verstärker	X-Achse	Y-Achse	Z-Achse	W-Achse
16 Bit DAC ± 10 Volt	DACB1-4	Ref+ or V+	31	32	33	34
Analog Ground	AGNDB (1-4)	Ref- or GND	30	30	30	30
16 Bit DAC ± 10 Volt	DACA1-4	Ref+ or V+	65	66	67	68
Analog Ground	AGNDA (1-4)	Ref- or GND	64	64	64	64
PWM 50/50	PWM Magnitude nA	PWM Phase 1	1	4	7	10
PWM 50/50	PWM Magnitude nB	PWM Phase 2	35	38	41	44
PWM 50/50	PWM Magnitude nC	PWM Phase 3	2	5	8	11
Hall Sensor	Hall A (1-4)	To motor Halls	13	15	17	19
Hall Sensor	Hall B (1-4)	To motor Halls	47	49	51	53
Hall Sensor	Hall C (1-4)	To motor Halls	14	16	18	20
Ground	Ground	GND	48	50	52	54
Ground	Ground	GND	36	37	39	40

Mikroschrittmotormodus

POSYS® 94x/96x/97x/98x
POSYS® 192x

(setzt OPTION CON J5 voraus)

GP CON J6, 100-poliger Stecker

Motorausgabe-methode	Signal	Verbindung zum Verstärker	X-Achse	Y-Achse	Z-Achse	W-Achse
16 Bit DAC ± 10 Volt	DACV 1-4	Ref+ or V+	17	34	67	84
Analog Ground	AGND (1-4)	Ref- or GND	18	35	68	85

In diesem Modus sind DACV1-4 und AGND (1-4) doppelt vorhanden (on GP CON and OPTION CON), aber nur das eine oder andere muss angeschlossen werden. Der Anwender kann entscheiden welche Anschlüsse für seine Applikation am bequemsten zu verwenden sind.

OPTION CON J5, 68-poliger Stecker

Motorausgabe-methode	Signal	Verbindung zum Verstärker	X-Achse	Y-Achse	Z-Achse	W-Achse
16 Bit DAC ± 10 Volt	DACB 1-4	Ref+ or V+	31	32	33	34
Analog Ground	AGNDB (1-4)	Ref- or GND	30	30	30	30
16 Bit DAC ± 10 Volt	DACVA1-4	Ref+ or V+	65	66	67	68
Analog Ground	AGNDA (1-4)	Ref- or GND	64	64	64	64
PWM Magnitude	PWM Magnitude nA	PWM Magnitude	1	4	7	10
PWM Sign	PWM Sign nA	PWM Direction	3	6	9	12
PWM Magnitude	PWM Magnitude nB	PWM Magnitude	35	38	41	44
PWM Sign	PWM Sign nB	PWM Direction	2	5	8	11
PWM 50/50	PWM Magnitude nA	PWM Phase 1	1	4	7	10
PWM 50/50	PWM Magnitude nB	PWM Phase 2	35	38	41	44
PWM 50/50	PWM Magnitude nC	PWM Phase 3	2	5	8	11

In den Tabellen nicht spezifisch erwähnte Anschlüsse sind über alle Modelle für J5 und J6 ansonsten gleich.

¤

Elektrische Spezifikationen

Elektrische Spezifikationen

Elektrische Spezifikationen

POSYS 7xx

Dimensions:	ISA-bus Karte, halbe Baugröße; (122 x 160 mm)
Storage temperature:	-40° bis +125° C
Operating temperature:	0° bis +70° C (standard commercial version) optional: -40° bis +85° C (extended temperature range)
Power supply voltage:	4.8 V bis 5.25 V Betriebsspannung, 1 A (DAC Ausgänge ein): 0.6 A (DAC ausgä aus)
Supply voltage limits:	-0.3 V bis +7 V
Analog output range:	-10.0 V bis +10.0 V, ±5 mA/axis, ±20 mA max./axis
Analog input range:	0 - 2.5 V
Digital outputs drive capacity:	DC output source or sink current: +/- 50 mA
Under voltage detection:	The under voltage supervisory device used is a MC P120T-475I. Under voltage specs are Min: 4.50 V, Max: 4.75 V, Typical: 4.625 V.

POSYS 8xx-B

Dimensions:	standard PC/104 Karte; (96mm x 115.5mm)
Storage temperature:	-40° bis +125° C
Operating temperature:	0° bis +70° C (standard commercial version) optional: -40° bis +85° C (extended temperature range)
Power requirement:	4.8 V bis 5.25 V Betriebsspannung, 1 A (DAC Ausgänge ein): DAC Outputs ausgeschaltet: 0.6 A
Supply voltage limits:	-0.3 V bis +7 V
Analog output range:	-10.0 V bis +10.0 V, ±5 mA/axis, ±20 mA max./axis
Analog input range:	0 - 4.096 V
Digital outputs drive capacity:	DC output source or sink current: +/- 50 mA
Under voltage detection:	The under voltage supervisory device used is a MC P120T-475I. Under voltage specs are Min: 4.50 V, Max: 4.75 V, Typical: 4.625 V.

POSYS 9xx

Dimensions:	PCI-bus Karte, halbe Baugröße (175 mm x 112 mm)
Storage temperature:	-40° bis +125° C
Operating temperature:	0° bis +70° C (standard commercial version) optional: -40° bis +85° C (extended temperature range)
Power requirement:	4.8 V bis 5.25 V operating range, 1 Amp no load current: DAC Outputs ausgeschaltet: 0.6 A
Supply voltage limits:	-0.3 V bis +7 V
Analog output range:	-10.0 V bis +10.0 V, ±5 mA/axis, ±20 mA max./axis
Analog input range:	0 - 4.096 V
Digital outputs drive capacity:	DC output source or sink current: +/- 50 mA
Under voltage detection:	The under voltage supervisory device used is a MC P120T-475I. Under voltage specs are Min: 4.50 V, Max: 4.75 V, Typical: 4.625 V.

POSYS 19xx

Dimensionen	PCI-Bus Karte; halbe Baugröße (171.45 x 111.76 mm)
Lagertemperatur	-40° bis +125° C

Betriebstemperatur	0° bis +70° C (standard commercial version) optional: -40° bis +85° C (extended temperature range)
Spannungs- und Stromversorgung	4.8V bis 5.25V Betriebsbereich, 1 Amp DAC Outputs ausgeschaltet: 0.6 A
Min./Max. Spannung	-0.3V bis +7V
Motorausgangssignalspannung und Strom	-10.0V bis +10.0V, ±5 mA min./Achse, ±20 mA max./Achse
Analoger Spannungseingangsbereich	0V bis 2.048V
Ausgangsstrom für digitale Ausgänge	± 50 mA
Unterspannungserkennung	Verwendetes Gerät zur Erkennung von Unterspannung ist: MC P120T-475I. Spezifikationen für Unterspannung: Min: 4.50 V, Max: 4.75 V, typisch: 4.625 V.

POSYS 18xx

Dimensions:	standard PC/104 Karte; (96mm x 115.5mm)
Storage temperature:	-40° bis +125° C
Operating temperature:	0° bis +70° C (standard commercial version) optional: -40° bis +85° C (extended temperature range)
Power requirement:	4.8 V bis 5.25 V Betriebsspannung, 1 A (DAC Ausgänge ein): DAC Outputs ausgeschaltet: 0.6 A
Supply voltage limits:	-0.3 V bis +7 V
Analog output range:	-10.0 V bis +10.0 V, ±5 mA/axis, ±20 mA max./axis
Analog input range:	0 - 2.048 V
Digital outputs drive capacity:	DC output source or sink current: +/- 50 mA
Under voltage detection:	The under voltage supervisory device used is a MC P120T-475I. Under voltage specs are Min: 4.50 V, Max: 4.75 V, Typical: 4.625 V.

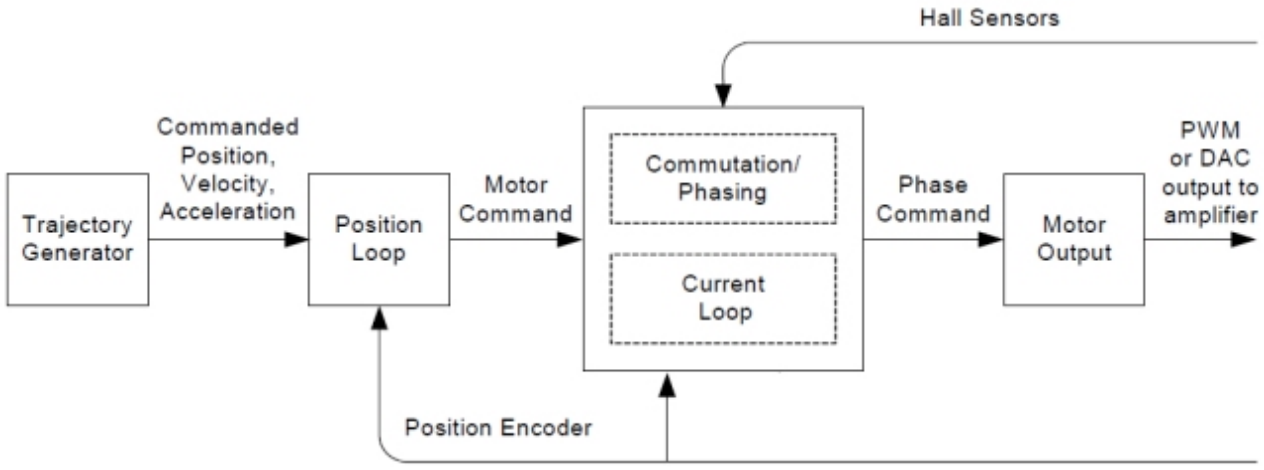
¤

Usability

Control Modules for POSYS® 1800/1900 mit Firmware Version >2.0

Control Flow Überblick

Die folgende Grafik gibt einen Überblick über die C ontrol Flow Funktionen der POSYS® Positioniersteuerkarten. Es zeigt wie ein Motorkommandosignal generiert wird, beginnend mit der Profilauswahl und endend mit der eigentlichen Ausgabe des Motorausgabemoduls welches verstärker-kompatible Sollwertvorgaben ausgibt. Abhängig vom Produkt und verwendetem Motortyp sind manche Module nicht verwendbar. z.B. sind ist im Schrittmotormodus die Auswahl des Position PID Loops nicht möglich.



Control Flow Überblick

Jedes der Hauptblöcke im C ontrol Flow Überblick ist assoziiert mit einem Modul. Die folgende Tabelle gibt einen kurzen Überblick der einzelnen Module und ihrer Bedeutung.

Modulname	Funktion
Trajectory Generator	Dieses Modul akzeptiert benutzer-definierte Eingaben und generiert daraus das Bahnprofil
Position Loop	Dieses Modul wird nur in Verbindung mit Servomotoren eingesetzt. Die Sollposition (momentane sbsolute gewünschte Position) und die Istposition (momentane tatsächliche Position) werden eingegeben und miteinander verglichen, woraus sich der Schleppabstandsfehler errechnet und dann durch den PID-Filter evtl. zusammen mit aktivierten Biquad-Filtern geschickt und generiert daraus eine Sollwertvorgabe für die Motorsignalausgabe.
Commutation/Phasing	Dieses Modul wird in Verbindung mit mehrphasigen Motoren verwendet (bürstenlose Servomotoren, Mikroschrittmotoren) und generiert für jede Phase die benötigte Sollwertvorgabe.
Motor Output	Dieses Modul übergibt das gewünschte Motorphasenkommando und erzeugt die entsprechenden Signale für das ausgewählte Ausgabeformat.

Jedes dieser Module ist in den nachfolgenden Kapiteln beschrieben. Darüberhinaus bieten die POSYS® Positioniersteuerungen zusätzliche Funktionalitäten wie z.B. Breakpoints, racing, SPS-ähnliche Signalkontrolle usw. Diese Funktionen sind für alle Modelle gleichermaßen verfügbar und werden ebenso in späteren Kapiteln näher beschrieben.

¤

Aktivieren und Deaktivieren von Control Modules

Beim Setup und im Betrieb kann es durchaus dem Wunsch entsprechen bestimmte C ontrol Module zu aktivieren oder deaktivieren. Dies erfolgt mit dem Kommando SetOperatingMode. Um den mit diesem Kommando gesetzten Status zurück zu lesen wird das Kommando GetOperatingMode verwendet. Generell gesagt: der Motion C ontroller überspringt sämtliche mit dem Modul assoziierten Kalkulationen und die Eingabe vom vorhergehenden Modul wird direkt weitergeleitet zum nachfolgenden Modul ohne Änderungen.

Die folgende Tabelle erläutert welche Module deaktiviert und aktiviert werden können und beschreibt typische Umstände unter denen dies sinnvoll erscheint. Zusätzliche zu der Funktionalität einzelne Module zu aktivieren/deaktivieren können ganze Achsen völlig stillgelegt werden. Bitte beachten Sie dass das Koomutierungs- und Phasenmodul durch den Anwender nicht aktiviert oder deaktiviert werden kann. Falls ein bürstenloser Servomotor oder Mikroschrittmotor ausgewählt wurde, ist dieses Modul immer aktiviert.

Modul	Beschreibung	Typische Anwendung
Trajectory Generator	Falls deaktiviert verbleibt die Sollposition bei seinem gegenwärtigen Wert	Dieses Modul wird für gewöhnlich nicht manuell deaktiviert. Für die Bahnkontrolle welches einen sofortigen Stopp erfodert, würde das Kommando SetStopMode mit dem Argument *Abrupt Stop* stattdessen verwendet.
Position Loop	Wird nur in Verbindung mit Servomotoren verwendet. Falls deaktiviert, gibt das Modul Ausgaben aus zwei Quellen abhängig davon ob das Modul aktiv oder deaktiv ist. *Trajectory Module enabled*: Falls aktiviert, wird der Position Loop übersprungen und die Ausgabe des Profile Generators direkt in das nachfolgende Modul eingegeben. *Trajectory Module disabled*: Falls deaktiviert, kommt die Ausgabe vom Motor Command Register, welches manuell eingegeben werden kann mit dem Kommando SetMotorCommand. *Im Kapitel "Deaktivieren und Aktivieren des Position Loop Modules" werden hierzu weitere Informationen bereitgestellt.*	Das Deaktivieren dieses Modules während der Trajectory Generator aktiviert ist, ist sinnvoll falls spannungs- oder stromporpotional agierende Rückmeldeeinheiten (z.B. Galvanometer) verwendet werden. Es kann auch sinnvoll sein für die Verstärkerkalibrierung falls benötigt. Zusammen mit deaktiviertem Trajectory Generator und deaktiviertem Position Loop Module ist dies sinnvoll für Verstärker- und Motorkalibrierung.
Motorausgabe	Die Deaktivierung dieses Moduls hat zur Folge, dass alle Werte für die Motor-spezifischen Ausgaben auf 0 (Null) gesetzt werden. Der aktuelle Zustand der assoziierten Motorausgabesignale wird von der gewählten Ausgabemethode abhängen (PWM 50/0, PWM s/m, DAC)	Die Deaktivierung der Motorsignalausgabe ist sinnvoll unter verschiedenen sicherheits-relevanten Bedingungen, oder für die Verstärkerkalibrierung

Zusätzlich zur manuellen Deaktivierung verschiedener Kontrollmodule gibt es eine Reihe Bedingungen unter denen die automatische Deaktivierung sinnvoll sein kann, wie z.B. Ereignis-bedingte oder durch Breakpoints bestimmte Bedingungen.

GetOperatingMode liefert den programmierten Status zurück der mit dem Kommando SetOperatingMode den gewünschten Operationsmodus für normale Bedingungen setzt. Jedoch kann dies vom aktuellen Operationsmodus aufgrund oben angeführter Gründe abweichen. Um den aktuellen Status abzufragen wird das Kommando GetActiveOperatingMode verwendet.

¤

Deaktivieren und Aktivieren des Position Loop Modules

nur verfügbar für POSYS® 1800/1900 mit Motion Control Firmwarev ersion >2.x

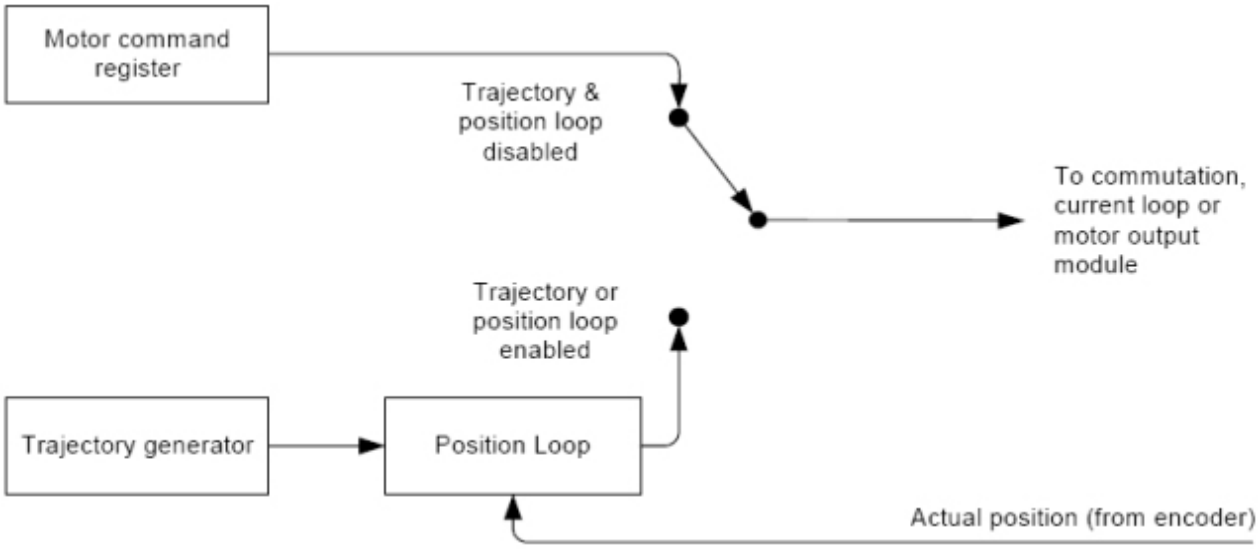
Es gibt verschiedene Gründe warum es wünschenswert sein kann das Position Loop Modul zu deaktivieren (siehe Aktivieren und Deaktivieren von Kontrollmodulen). Zusätzlich exisiteren Ereignis-bedingte Aktionen die als Ergebnis die Deaktivierung des Position Loop Modules zur Folge haben (siehe SetEventAction Processing).

Falls das Position Loop Modul deaktiviert wurde, kann das auf den allgemeinen Kontrollfluss zweierlei Auswirkungen haben, abhängig davon ob auch der Bahngenerator aktiv oder deaktiv ist.

Bahngenerator deaktiviert: Falls der Bahngenerator deaktiviert ist, wenn die Regelschleife deaktiviert wird, wird die Ausgabe des Lagereglermodules ein 16-Bit Wort sein, der von einem programmierbaren Motorbefehlsregister abgeleitet, mit dem Kommando SetMotorCommand programmiert wurde und mit dem Kommando GetMotorCommand gelesen werden kann.

Bahngenerator aktiviert: Falls der Bahngenerator aktiviert ist, wenn die Regelschleife deaktiviert wird, wird das Lagereglermodul dann umgangen, und der Ausgangswert des Bahngenerators wird der Motorbefehlswert sein.

Egal ob das Lagereglermodul aktiviert oder deaktiviert ist, der eigentliche wirksame Motorbefehlswert kann mit dem Kommando GetActiveMotorC ommand abgefragt werden. Dies ist der in dem Moment tatsächlich an die nachfolgenden Module und den Verstärker übergebene Wert für die Motorausgabe.


Motor command register
Trajectory & position loop disabled
To commutation, current loop or motor output module
Trajectory or position loop enabled
Trajectory generator
Position Loop
Actual position (from encoder)

Ein vorher deaktiviertes Lagereglermodul kann auf verschiedene Arten wieder freigegeben werden. Falls das Modul mit dem Kommando SetOperatingMode daktiviert wurde, kann ein neuerliches SetOperatingMode Kommando es wieder aktivieren. Wenn das Lagereglermodul als Teil einer automatischen Ereignis-bezogenen Aktion (siehe SetEventAction Processing, für Details) deaktiviert wurde, verwendet man dann das Kommando RestoreOperatingMode um das Modul wieder zu aktivieren.

Ungeachtet wie das Modul wieder freigegeben wird, zu dem Zeitpunkt zu dem die Reaktivierung initiiert wird, werden bestimmte Vorgänge ausgeführt um unerwartetes Verhalten der Achse zu vermeiden. Im besonderen werden alle Lagereglerzustandsgrößen auf null (0) gesetzt.

¤

Position Loop Werte lesen

nur verfügbar für POSYS® 1800/1900 mit Motion Control Firmwarev ersion >2.x

Es gibt eine Reihe Kommandos die verwendet werden können, verschiedene Werte für das Position Loop auszulesen. Unter anderem sind dies GetCommandedPosition um die Sollposition zu lesen GetActualPosition um die tatsächliche aktuelle Position zu lesen die vom Enkoder zurückgeliefert wird GetPositionError um den Schleppabstandsfehler (die Differenz der beiden vorausgegangenen Werte) zu erfahren und GetActiveMotorCommand um die aktuelle Motorausgabe des Position Loop Modules zu lesen. Diese Werte sind nicht nur lesbar sondern auch für das Tracing auswählbar.

Neben diesen Registern, um die Abstimmung weiter zu erleichtern, exisitieren eine Anzahl weiterer Lagereglerwerte, die zurückgelesen als auch nachverfolgt werden können. Das Kommando zum zurücklesen lautet GetPositionLoopValue

Die Variablen innerhalb des Position Loops sind in der nachfolgenden Tabelle gelistet und zusammengefaßt.

Variablenname	Funktion
Integrator Sum	Dieses Register liefert die Summe des Integrator's für den Position Loop
Integral Contribution	Dieses Register hält den umfassenden Beitrag vom Integrator zum Position-PID-Loop.
Derivative	Dieses Register hält den Lagefehlerableitungswert. Das heißt der Unterschied zwischen dem gegenwärtigen Lagefehler und dem vorhergehenden Lagefehler.
BiQuad1 Input	Dieses Register beinhaltet den Eingabewert für das BiQuadfilter #1
BiQuad2 Input	Dieses Register beinhaltet den Eingabewert für das BiQuadfilter #2

¤

Profilerstellung

Bahnen, Profile und Parameter

Bahnen, Profile, und Parameter

Der Bahngenerator kalkuliert die aktuelle Position, Geschwindigkeit und Beschleunigung zu einer gegebenen Zeit. Diese Werte werden die Sollwerte genannt. Während einer Bewegungssequenz werden sich einige oder alle dieser Parameter kontinuierlich ändern. Sobald die Bewegungssequenz vollständig ist, werden diese Parameter aktuell bleiben, bis eine neue Bewegung begonnen wird und mit ihr neue Parameter übergeben werden.

Um die aktuellen Sollprofilwerte abzufragen, werden die Befehle GetCommandedPosition, GetCommandedVelocity, und GetCommandedAcceleration benutzt.

Durch das ganze Handbuch hindurch werden verschiedene mnemotechnische Kommandos gezeigt um die Verwendung der POSYS® zu erklären oder spezifische Beispiele zu liefern. Der Programmierabschnitt liefert mehr Informationen über Befehle, Nomenklatur und Syntax.

Das spezifische Profil, dass die POSYS® einrichtet, hängt von einigen Faktoren einschließlich des aktuellen Profilmodus, der aktuellen Profilparameter, und anderer Systemzustände ab, wie z.B. ob ein Bewegungsstopp ausgelöst worden ist. Sechs Bahnprofilmodi werden unterstützt:

- S-Kurve Punkt-zu-Punkt.
- Trapezförmig (linear) Punkt-zu-Punkt.
- Trapezförmig Punkt-zu-Punkt mit Smoothingfaktor (Option).
- Geschwindigkeitsregelung.
- Electronic Gearing (Folge-/Leitbetrieb).
- External Profile Mode

Die Verwendung dieser Profilmodi wird ausführlich in nachfolgenden Abschnitten erklärt. Der verwendete Befehl, den Profilmodus zu selektieren, ist SetProfileMode. Der Befehl GetProfileMode liefert den programmierten Profilmodus zurück.

Der Profilmodus kann unabhängig für jede Achse programmiert werden. Zum Beispiel Achse #1 kann im trapezförmigem Modus sein während Achse #2 für den S-Kurvenmodus programmiert wurde.

Mit einer Ausnahme kann die POSYS® von einem Profil in das Andere schalten, während eine Achse in Bewegung ist. Die Ausnahme: um in den S-Kurven Punkt-zu-Punkt Modus zu schalten muß die Achse stillstehen. Es ist auch nicht möglich, während eine Bewegung noch abläuft, vom S-Kurvenmodus in einen anderen Modus zu schalten.

¤

Definition der Parameter

Definition der Parameter

Die POSYS® sendet und empfängt Bahnprofildaten und verwendet dabei eine Parameterübergabe nach Fixpunktdarstellung. In anderen Worten es werden eine feste Anzahl Bits verwendet um den Integerwert einer Realzahl darzustellen.

Folgende 3 Formate verwendet die POSYS®:

Format	Wortgröße	Bereich	Beschreibung
32.0	32 Bit	-2.147.483.648 bis +2.147.483.647 Zähler	Unity Scaling. Dieses Format verwendet nur Integerzahlen.
16.16	32 Bit	-32.768 bis 32,767	Verwendet 1/216 Skalierung. Die POSYS® erwartet eine 32-B

Format	Wortgröße	Bereich	Beschreibung
		+65.535/65.536	Nummer welches mit einem Faktor von 65.535 skaliert wurde z.B. um eine Geschwindigkeit von 2,75 zu definieren, muss 2, mit 65.535 multipliziert werden und das Resultat als 32-Bit Integer an die POSYS® geschickt werden (180.224 dez. oder 2c000h).
0.32	32 Bit	2.147.483.648/ 4.294.967.296 bis +2.147.483.647/ 4.294.967.296	Verwendet 1/232 Skalierung. Die POSYS® erwartet eine 32-B Nummer welches mit einem Faktor von 4.294.967.296 (232) skaliert wurde. z.B. um einen Wert von 0,0075 zu übergeben, muss 0,0075 mit 4.294.967.296 (232) multipliziert werden un das Resultat als 32-Bit Integer an die POSYS® werden (32.212.256 dez. oder 1eb8520h).

Die Umwandlung der Daten in die benötigten Formate wird bereits durch die mitgelieferte DLL erledigt.

¤

Trapezfömiger Punkt-zu-Punkt Profil

Trapezförmiger Punkt-zu-Punkt Profil

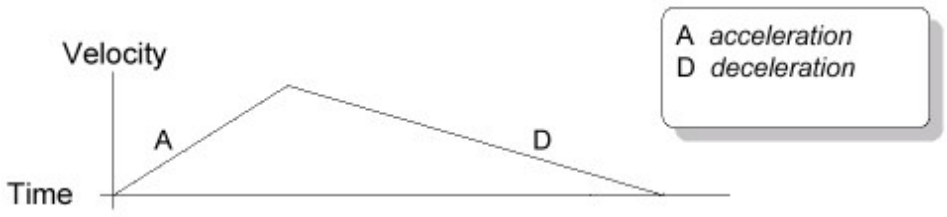
Die folgende Tabelle umfasst die spezifischen Profilparameter für den trapezförmigen Punkt-zu-Punkt Modus:

Profilparameter	Format	Wortgrösse	Bereich
Position	32.0	32 bits	-2.147.483.648 to +2.147.483.647 counts
Velocity	16.16	32 bits	0 to 32.767+65.535/65.536 counts/cycle
Acceleration	16.16	32 bits	0 to 32.767+65.535/65.536 counts/cycle²
Deceleration	16.16	32 bits	0 to 32.767+65.535/65.536 counts/cycle²

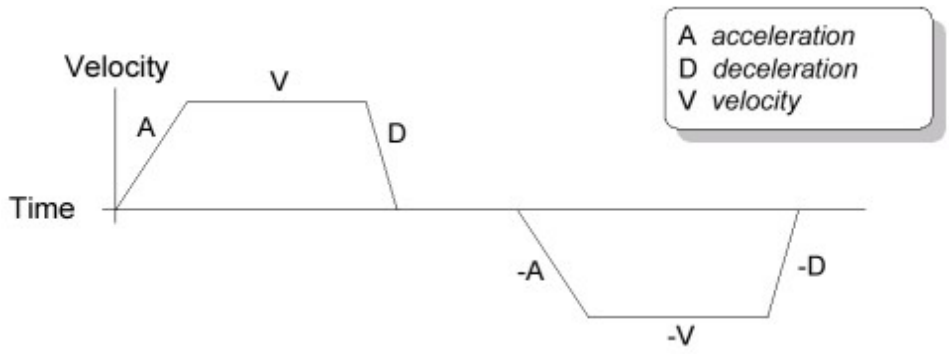
Die Kommandos SetPosition, SetVelocity, SetAcceleration, und SetDeceleration laden diese Werte. Die Befehle GetPosition, GetVelocity, GetAcceleration, und GetDeceleration liefern die programmierten Werte zurück.

Für dieses Profil errechnet der Host eine anfängliche positive und negative Beschleunigung, eine Geschwindigkeit, und eine Endposition. Das Profil bekommt seinen Namen aufgrund der Kurvenform (Figur 1-1b): die Achse beschleunigt linear (anhand des programmierten Beschleunigungswertes), bis es die programmierte Geschwindigkeit erreicht. Die Geschwindigkeit bleibt erhalten, bremst dann linear ab (den negativen Beschleunigungswert benutzend, falls eine programmiert wurde), bis es an der angegebenen Position stehen bleibt. Falls die negative Beschleunigung einsetzen muss ehe die Achse die programmierte Geschwindigkeit erreicht, wird das Profil kein konstantes Geschwindigkeitsteil ausweisen, und das Trapez wird ein Dreieck (Ziffer 1-1a).

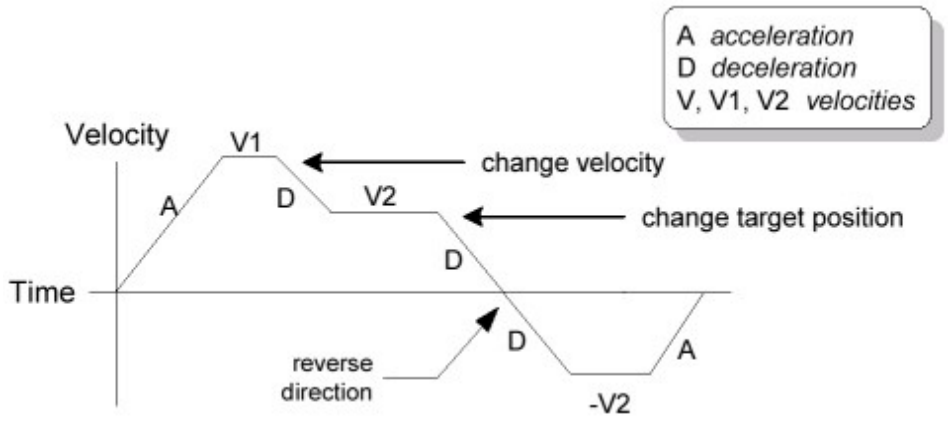
Trapezförmige Punkt-zu-Punkt Profile



Figur 1.1a



Figur 1.1b



Figur 1.1c

Die positiven und negativen Beschleunigungsrampen können symmetrisch (wenn positive Beschleunigung gleich negative Beschleunigung) oder asymmetrisch sein (wenn positive Beschleunigung nicht gleich der negativen Beschleunigung ist).

Der Beschleunigungsparameter wird immer am Anfang der Bewegungssequenz benutzt. Danach wird der Wert für die positive Beschleunigung in jene Richtung verwendet, und der Wert für die negative Beschleunigung wird für die entgegengesetzte Richtung eingesetzt. Falls keine Bewegungsparameter während der Bewegungssequenz verändert werden, dann wird der positive Beschleunigungswert verwendet bis die maximale Geschwindigkeit erreicht wurde und der negative Beschleunigungswert für die Abbremsrampe bis 0 erreicht wurde.

Es ist möglich, einen der Profilparameter zu verändern, während die Achse sich in diesem Profilmodus befindet. Der Profilgenerator wird immer versuchen, die Bewegung innerhalb der durch die Parameter vorgegebenen gesetzten Bedingungen auszuführen. Wird während der Bewegung die Endposition in solch einer Weise verändert, dass der Schleppabstand überschritten wird, wird die POSYS® mit Rampe bis zum Stopp abbremsen und dann in entgegengesetzte Richtung beschleunigen, um sich zu der neuen angegebenen Position zu bewegen.

Anmerkung: Bitte beachten Sie, dass die Richtung der negativen und positiven Beschleunigungswerte zu Beginn der Bewegung festgesetzt wurden. Das heisst dass der negative Beschleunigungswert für die Beschleunigung in entgegengesetzte Richtung verwendet wird wenn der endgültige Zielpunkt überschritten wurde und der Motor zurückfahren muss um diese Position zu erreichen. Dies wird in Figur 1.1c veranschaulicht.

Falls keine negative Beschleunigung programmiert wurde, oder = 0 ist, dann wird der Wert verwendet der für die positive Beschleunigung programmiert wurde.

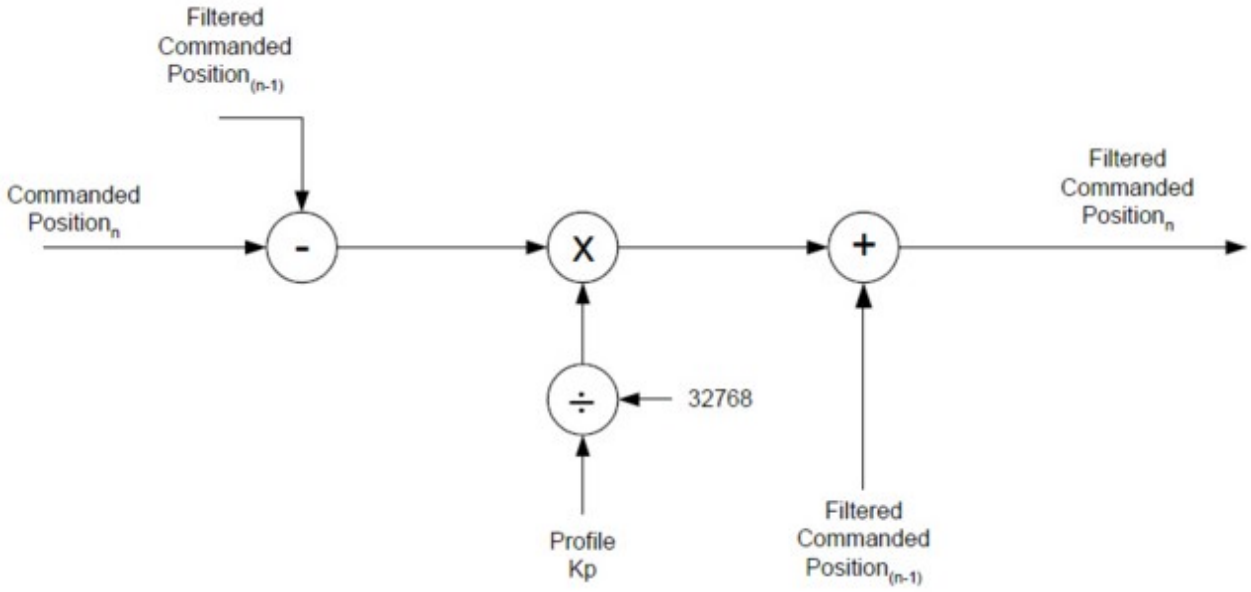
¤

Smoothing Factor

Filtered Trapezoidal Profile

Der Smoothing Factor ist eine Option und Erweiterung des Standard Linear Profils welches ein zusätzliches Filter zur Verfügung stellt. Weil es auf dem bestehenden linear Modus basiert sind Änderungen der Profilfunktionen wie Position, positive/negative Beschleunigung, Geschwindigkeit und assymetrische Bahngenerierung jederzeit möglich.

Dieses modifizierte Trapezprofil erzeugt ein S-Kurven ähnliches Geschwindigkeitsprofil und erlaubt "On-the-fly" Änderungen aller Bahn-relevanten Parameter. Der aktuelle Filter ist im folgenden Diagramm abgebildet.



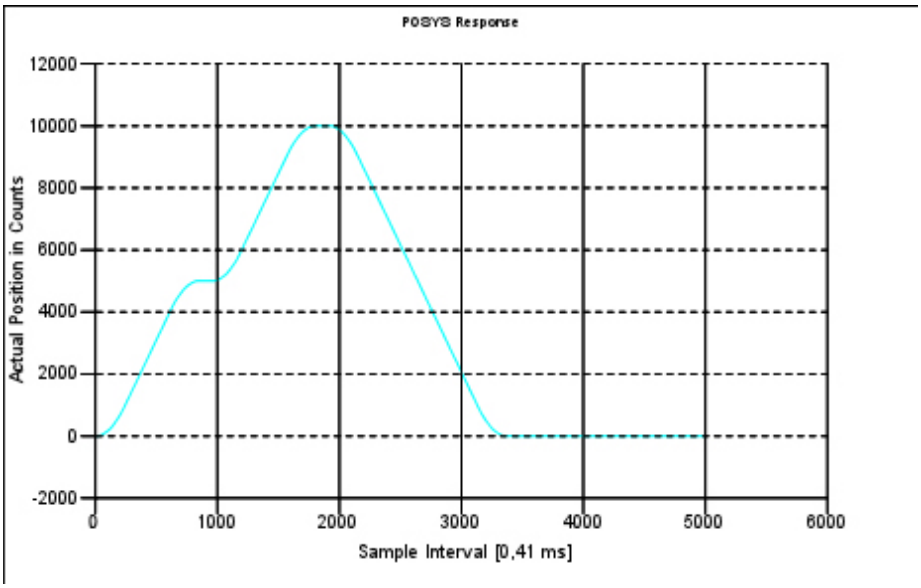
Der "ProfileKp" ist ein proportional wirkender Verstärkerparameter mit einem Einstellbereich zwischen 0 und 32767. Wenn auf 32767 gesetzt wird keine gefilterte Ausgabe erzeugt und ist gleich dem Wert 0. Für Werte niedriger als 32767 vergrößert sich entsprechend der Filter.

Sollgeschwindigkeit und -beschleunigung, die vom PID-Filter für die Feedforward Einheiten verwendet werden, werden von der gefilterten Bahn-Generierung hergeleitet.

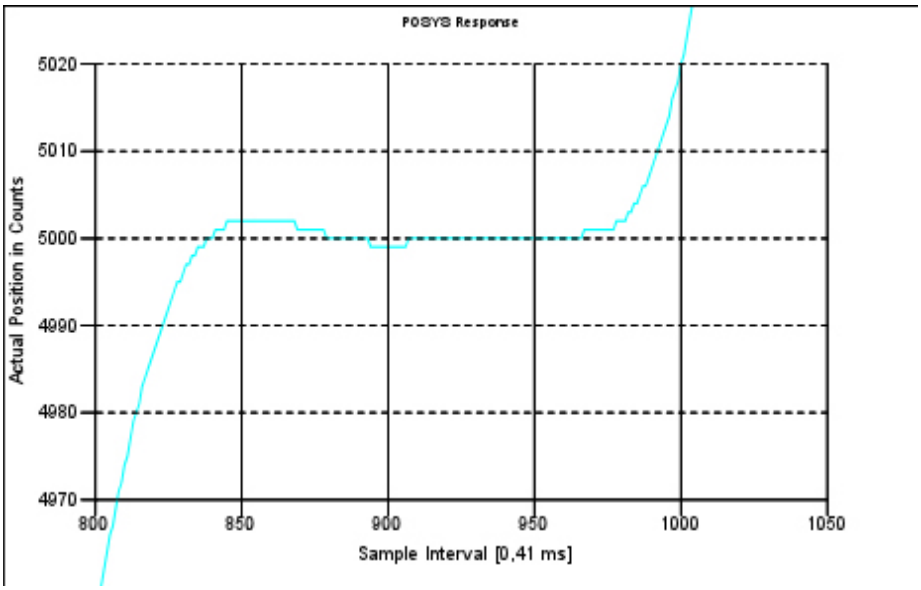
Der Standardwert für "ProfileKp" nach einem Reset oder nach dem Einschalten ist 0x7fff.

Die nachfolgenden Bilder zeigen die Veränderungen des Bewegungsablaufes ohne und dann mit Smoothing Faktor. Ohne eingestelltes Smoothing ist ein deutliches Überschiessen am Ende der jeweiligen Streckenabschnitte erkennbar. Mit Smoothing Faktor reduziert sich dieses Überschiessen auf Null.

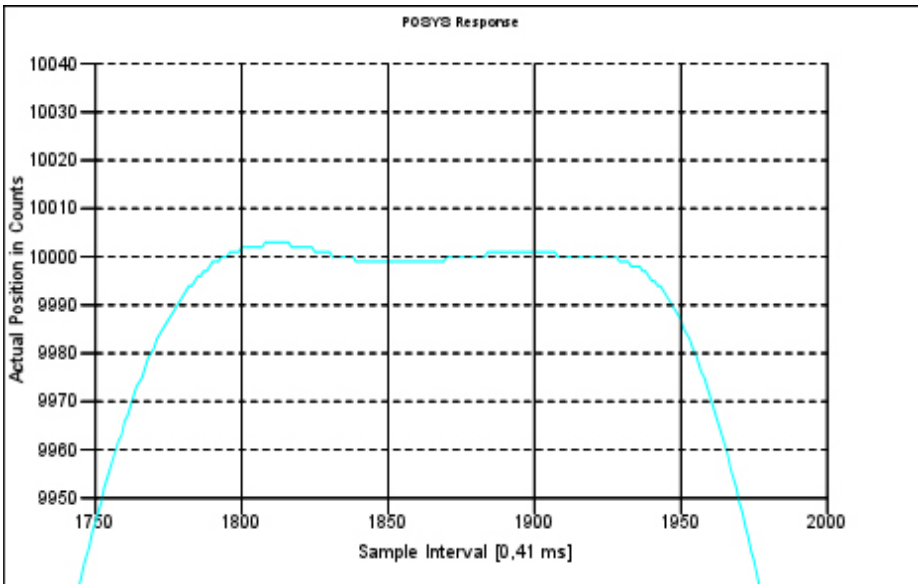
Die ersten 4 Bilder zeigen typisches Verhalten ohne Smoothing. Das erste Bild gibt einen Überblick über die gesamte Verfahrstrecke und die nächsten 3 Bilder zeigen vergrößert die Haltepunkte mit erneutem Anfahren (ausser 3. Bild das den Endpunkt markiert).



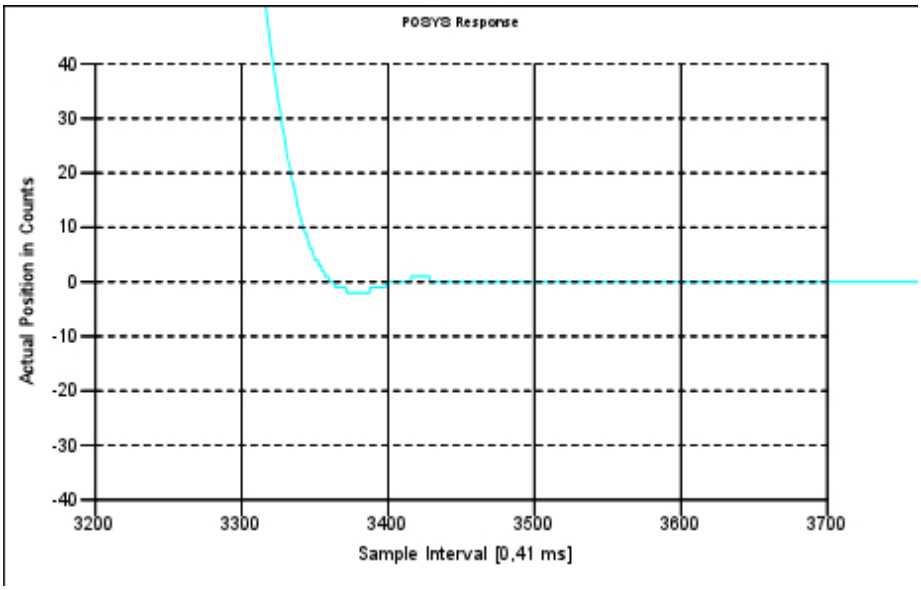
Gesamtverfahrstrecke ohne Smoothing Faktor



Erster Stopppunkt mit neuerlichem Anfahren nach 100 Millisekunden Stillstand. Leichtes Über-/Unterschwingen ist zu erkennen

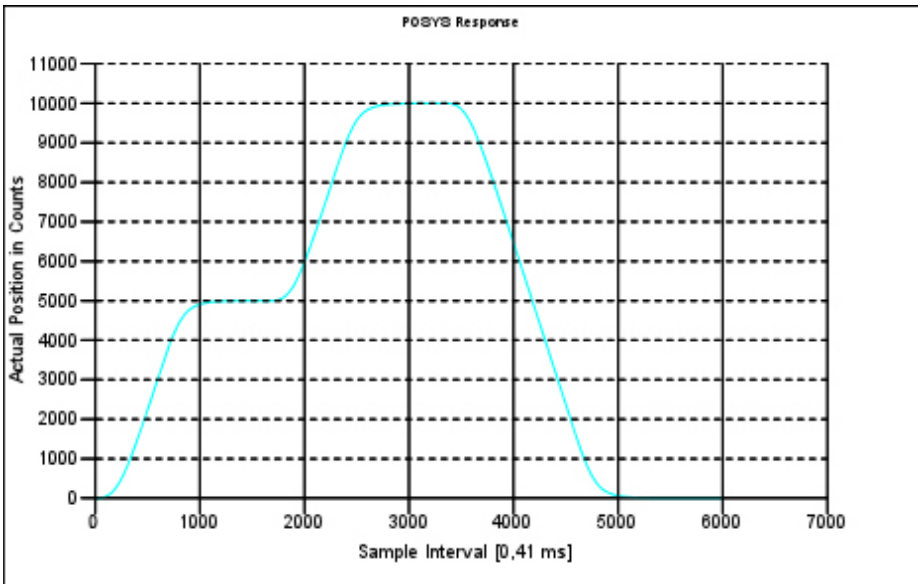


Zweiter Stopppunkt mit Über-/Unterschwingen und anschliessend negativer Fahrrichtung

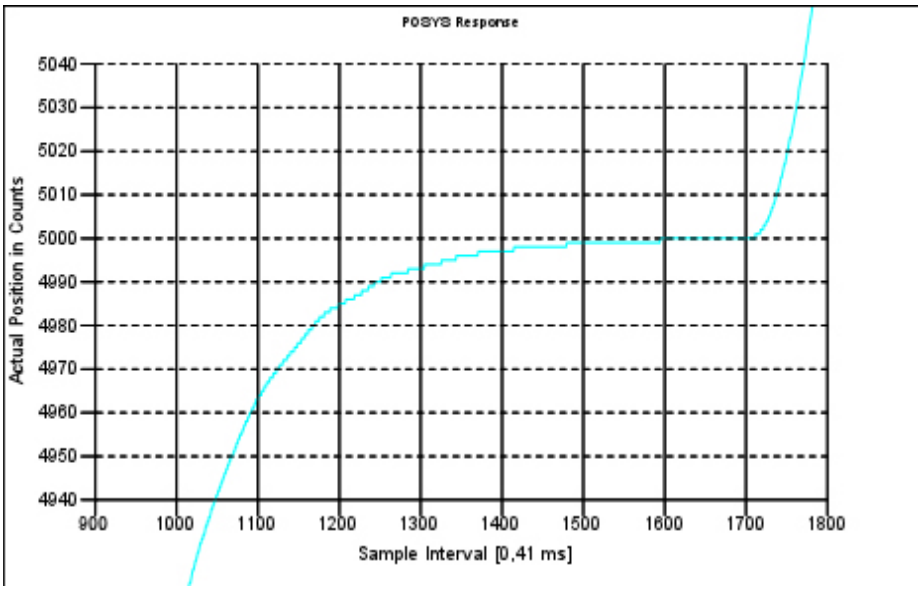


Endpunkt erreicht mit neuerlichem Über-/Unterschwingen

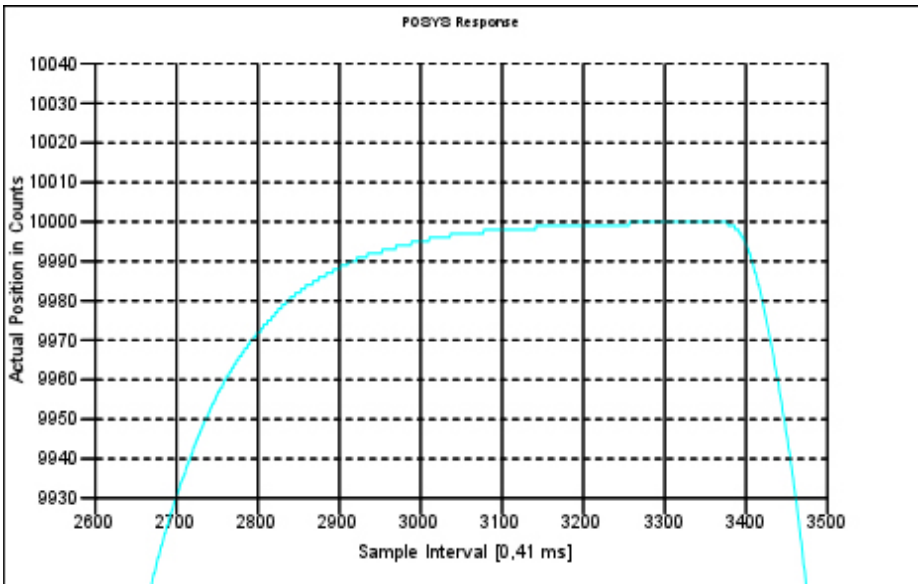
Die nächsten 4 Bilder zeigen den gleichen Bewegungsablauf mit eingestelltem Smoothing Faktor. Der Effekt der sich einstellt ist ähnlich dem S-Kurvenverlauf mit dem Unterschied, dass alle Parameter (Geschwindigkeit, Beschleunigung, Position) wie im Trapezmodus weiterhin "on-the-fly" veränderbar bleiben. Im Vergleich verlängert sich aber auch der gesamte zeitliche Ablauf.



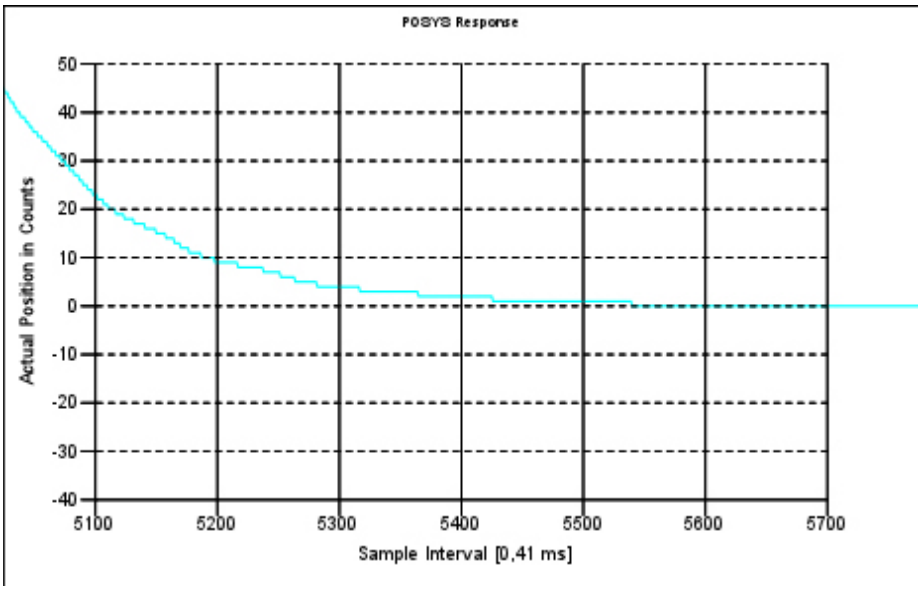
Gesamtverfahrstrecke mit Smoothing Faktor



Erster Stopppunkt mit neuerlichem Anfahren nach 100 Millisekunden Stillstand. Kein Über-/Unterschwingen mehr zu erkennen



Zweiter Stopppunkt ohne Über-/Unterschwingen und anschliessend negativer Fahrrichtung



Endpunkt erreicht ohne Über-/Unterschwingen

Alle Anzeigen in den Abbildungen spiegeln die aktuellen Positionen wider.

Set/GetProfileKp

¤

S-Kurven Punkt-zu-Punkt Profil

S-Kurven Punkt-zu-Punkt Profil

Die folgende Tabelle fasst die host-spezifischen Profilparameter zusammen für den S-Kurven Punkt-zu-Punkt Modus:

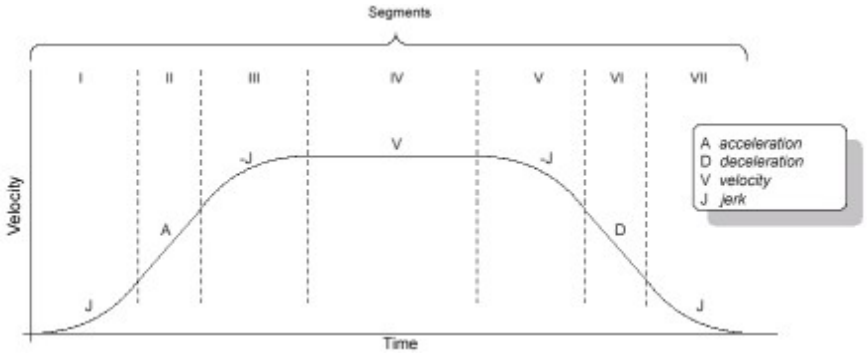
Profilparameter	Format	Bereich
Position	32 Bit	-2.147.483.648 to +2.147.483.647
Velocity	32 Bit	0 to 32.767+65.535/65.536 counts/cycle

Profilparameter	Format	Bereich
Acceleration	32 Bit	0 to 32.767+65.535/65.536 counts/cycle²
Deceleration	32 Bit	0 to 32.767+65.535/65.536 counts/cycle²
Jerk	32 Bit	0 to 2.147.483.647/4.294.967.296 counts/cycle³

Die Kommandos SetPosition, SetVelocity, SetAcceleration, SetDeceleration, und SetJerk laden die jeweiligen Werte. Die Befehle GetPosition, GetVelocity, GetAcceleration, GetDeceleration, und GetJerk liefern die programmierten Werte zurück. Das S-Kurven Punkt-zu-Punkt Profil fügt eine Begrenzung zum Verhältnis der Änderung der Beschleunigung hinzu, verglichen mit der Basis des Trapezprofils. Ein neuer Parameter (Jerk) spezifiziert die maximale Änderung der Beschleunigung innerhalb eines Updatezyklus.

In diesem Profilmodus wächst die Beschleunigung allmählich von 0 zu dem programmierten Beschleunigungswert an, dann vermindert sich die Beschleunigung im gleichen Verhältnis, bis es 0 mit der programmierten Geschwindigkeit erreicht. Die gleiche Sequenz wird ausgeführt um die Endposition zu erreichen.

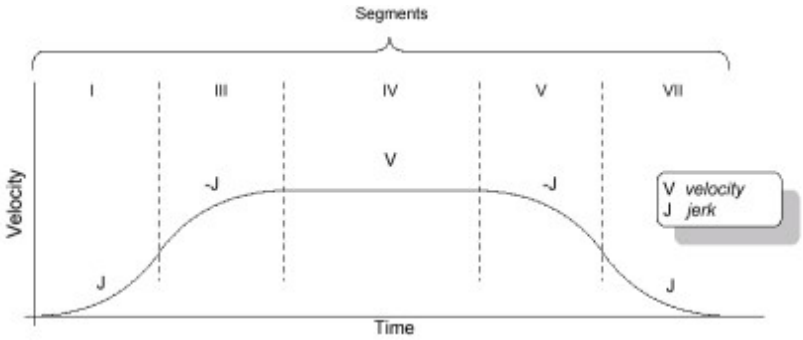
Im S-Kurven Profilmodus, muss der gleiche Wert sowohl für Beschleunigungs- als auch für die Abbremsrampe benutzt werden. Asymmetrische Profile sind nicht erlaubt. Die ist nur möglich im trapezförmigen Profilmodus mit Smoothingfaktor (Option).



Figur 2-1. S-Kurven Profil

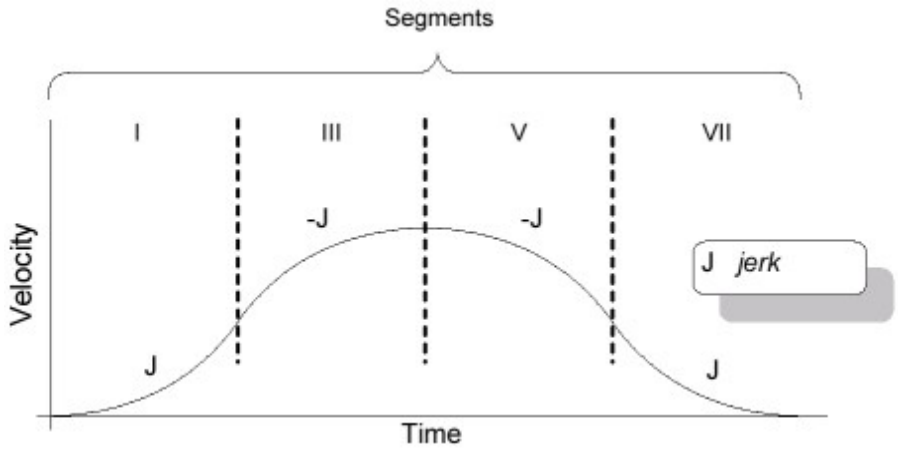
Figur 2-1 zeigt ein typisches S-Kurven Profil. In Segment I erhöht sich der Beschleunigungswert um den mit dem Befehl Jerk gesetzten Wert bis die maximale Beschleunigung erreicht wurde. Die Achse fährt fort, sich linear (jerk = 0) durch Segment II zu beschleunigen. Das Profil wendet dann im Segment III den negativen Wert des Jerks an um die Beschleunigung zu reduzieren. Im Segment IV verfährt die Achse jetzt mit maximaler (programmierter) Geschwindigkeit (V). Das Profil wird dann in einer dem Beschleunigungswert ähnlichen Weise abbremsen, indem in umgekehrter Richtung der negative Jerk verwendet wird, um zuerst die maximale Verlangsamung zu erreichen (A), und dann die Achse zu einem Halt an der Endposition zu bringen.

Eine S-Kurve mag vielleicht nicht alle Elemente der Figur 2-1 gezeigten Segmente enthalten. Zum Beispiel falls nicht die maximale Beschleunigung vor dem "Halbweg" in Richtung Endgeschwindigkeit oder Endposition erreicht werden kann, würde das Profil nicht die Segmente II und VI enthalten. Solch ein Profil wird in Figur 2-2 gezeigt.



Figur 2-2. S-Kurve erreicht nicht die maximale Beschleunigung

Ähnlicherweise, falls eine Position derart angegeben wird, dass die Endgeschwindigkeit nicht erreicht werden kann, wird es kein Segment IV, wie in Figur 2-3 gezeigt, geben. (Es ist auch möglich, dass es auch kein Segment II oder Segment VI gibt, je nachdem wo das Profil "gekürzt" wird.)



Figur 2-3. S-Kurve ohne maximaler Geschwindigkeitssegment

Im Gegensatz zum trapezförmigen Profilmodus, erlaubt der S-Kurven Profilmodus keine Änderungen an einem der Profilparameter während die Achse in Bewegung ist. Ebenfalls darf die Achse nicht in den S-Kurven Modus geschaltet werden während die Achse in Bewegung ist. Es ist allerdings erlaubt vom S-Kurven Modus zu einem anderen Profilmodus während der Bewegung zu wechseln. Falls der S-Kurven Modus für die Art der auszuführenden Bewegung notwendig ist und die Notwendigkeit besteht PID-Filter-, Geschwindigkeit- und/oder Beschleunigungswerte während der Bewegung zu ändern, dann ist es empfehlenswert den Trapezmodus mit "Smoothing-Factor" zu verwenden. Dieser Modus erlaubt ein S-Kurven Modus ähnliches Beschleunigungsverhalten mit weicheren Übergängen vom Beschleunigungssektor zum Geschwindigkeitssektor bzw. umgekehrt und wenn man zu einem STOP kommt. Die Weichheit der Übergänge ist programmierbar, aber nicht in dem Maße wie im S-Kurvenmodus.

¤

Geschwindigkeitsmodus

Die folgende Tabelle fasst die host-spezifischen Profilparameter zusammen für den Geschwindigkeitsmodus:

Profilparameter	Word size	Bereich
Velocity	32 Bit	0 bis 32.767+65.535/65.536 counts/cycle
Acceleration	32 Bit	0 bis 32.767+65.535/65.536 counts/cycle²
Deceleration	32 Bit	0 bis 32.767+65.535/65.536 counts/cycle²

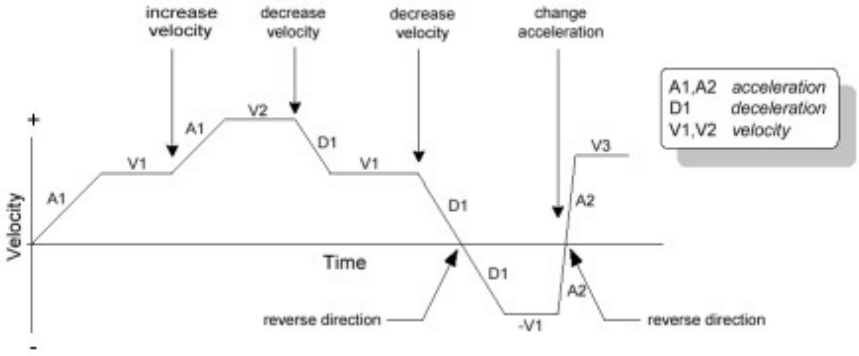
Die Kommandos SetVelocity, SetAcceleration, und SetDeceleration laden die jeweiligen Werte. Die Befehle GetVelocity, GetAcceleration, und GetDeceleration, liefern die programmierten Werte zurück.

Im Gegensatz zu den trapezförmigen und S-Kurven Profilmodi, welche die Endposition bestimmt welche Richtung positiv oder negativ wird, bestimmt das Vorzeichen im Geschwindigkeitsmodus die positive oder negative Richtung. Deswegen kann der Geschwindigkeitswert, der zur POSYS® gesandt wird, positive Werte (für eine positive Richtungsbewegung) oder negative Werte haben (für eine entgegengesetzte Richtungsbewegung). In diesem Profil wird keine Endposition angegeben. Die Bewegung wird gänzlich kontrolliert, indem man die positive/negative Beschleunigung und Geschwindigkeit verändert, während das Profil ausgeführt wird.

Die Bahn wird ausgeführt, indem man kontinuierlich die Achse mit dem angegebenen Wert beschleunigt, bis die Geschwindigkeit erreicht wird. Die Achse fängt an abzubremsen wenn eine neue Geschwindigkeit angegeben wird der einen kleineren Wert hat als die aktuelle Geschwindigkeit, oder ein anderes Vorzeichen hat als die aktuelle Richtung vorgibt.

Ein einfaches Geschwindigkeitsprofil sieht aus wie ein einfaches trapezförmiges Punkt-zu-Punkt Profil, wie in Figurr 1-1 dargestellt.

Figur 3-1 zeigt ein komplizierteres Profil, in dem Beides, die Geschwindigkeit als auch die Bewegungsrichtung zweimal wechseln.



Figur 3-1 Geschwindigkeitsprofil

Anmerkung: Im Geschwindigkeitsmodus ist die Achsenbewegung nicht an eine Endposition gebunden. Es liegt in der Verantwortung des Anwenders, Geschwindigkeits- und Beschleunigungswerte zu verwenden die einen sicheren Bewegungsablauf garantieren.

¤

Electronic Gear Profile

Electronic-Gear Profil

Die folgende Tabelle fasst die host-spezifischen Profilparameter zusammen für den Electronic-Gear Modus:

Profilparameter	Word size	Bereich
Gear ratio	32 Bit	-32.767 bis 32.767+65.535/65.536 counts/cycle
Master axis #	2 Bit	0-3*
Master source	1 Bit	2 Werte; aktuelle oder Sollposition (siehe unten für Details)

*für 2-Achsen Versionen 0-1. 1-Achsen Karten unterstützen kein Electronic-Gear Modus.

Die Kommandos SetGearRatio, und SetGearMaster laden die jeweiligen Werte. Die Befehle GetGearRatio und GetGearMaster liefern die programmierten Werte zurück.

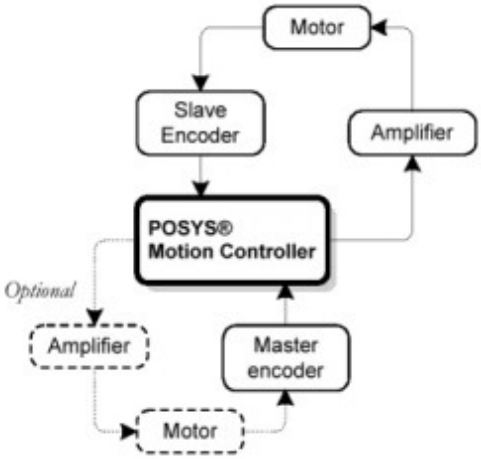
In diesem Profil gibt der Host drei Parameter an. Der Erste ist die „Master“-Achse #, die die Achse ist, die die Quellinformationen liefert um die „Slave“-Achse zu führen, die die Achse in Getriebemodus ist. Der Zweite ist die Gearsource die den Wert liefert nach der die Slave-Achse folgen soll. Es können entweder die Ist- oder Soll-Position sein. Der Dritte ist das Übersetzungsverhältnis, das angibt in welcher Richtung und mit welchem Verhältnis die Slave-Achse der Master-Achse folgen soll.

Ein positives Übersetzungsverhältnis bedeutet, dass wenn die aktuelle oder Soll-Position der Master-Achse sich in positiver Richtung erhöht, sich auch die Slave-Achse in positiver Richtung bewegt. Ein negatives Verhältnis hat den umgekehrten Effekt. Während die Position der Master-Achse sich erhöht verringert sich die Position der Slave-Achse.

Z.B., lassen Sie uns annehmen die Slave-Achse ist die Achse #0 (Achsen werden von 0 bis 3 bei einer 4-Achsen Karte gezählt) und die Master-Achse ist die Achse #3. Lassen Sie uns auch annehmen die Gearsource wird auf aktuelle Position mit einem Verhältnis von -½ gesetzt.

Dann für jeden positiven Enkoderzählwert von Achse 3, wird die Sollposition der Slave-Achse #0 sich um einen halben Enkoderwert in negativer Richtung verschieben. Ein negativer Enkoderzählwert der Master-Achse resultiert in einem halben positiven Enkoderzählwert für die Slave-Achse.

Die Figur 4-1 zeigt die Anordnung von Enkoder und Verstärker in einer typischen Electronic-Gearing Einstellung



Figur 4-1 POSYS® im Electronic-Gear Profilmodus

Der Electronic-Gearmodus erfordert, dass 2 Achsen definiert werden. Deswegen unterstützen die 1-Achsen Karten den Electronic-Gearmodus nicht. Falls die Gearsource der Master-Achse auf "Ist-Position" eingestellt wurde, ist es nicht notwendig einen Motor an dieser Achse zu betreiben. In der Hauptsache wird es als Enkodereingangszähler verwendet, z.B. im Open-Loop Modus oder als manuelle Einstellmöglichkeit für andere Achsen. Es ist allerdings auch möglich einen Motor zu verwenden, dabei muss dieser Achse ein anderer Profil zugewiesen werden. Der Effekt dieser Einstellung ist, dass beide Achsen, Master als auch Slave, mit dem selben Profil betrieben werden können, auch wenn der Slave mit einem anderen Übersetzungsverhältnis zum Master betrieben wird oder in eine andere Richtung verfährt. Für die Master-Achse ist es ohne Belang ob sie als solche oder als Einzelachse betrieben wird. Die "optionalen" Komponenten in Figur 4-1 illustrieren dieses Arrangement. Eine solche Konfiguration kann sinnvollerweise zur Verwendung von linear interpolierten Achsen eingesetzt werden.

Anmerkung:
Es ist möglich das Übersetzungsverhältnis während der Bewegung zu ändern, aber es liegt in der Verantwortung des Anwenders Übersetzungsverhältnisse zu wählen die einen sicheren Betrieb gewährleisten. Der Wechsel von einem kleinen zu einem großen Verhältnis kann eine Achse zu schnell an seine physikalische Endposition führen und zu Beschädigungen der Mechanik führen. Auch das Verletzungsrisiko kann sich aufgrund sich sehr schnell in der Geschwindigkeit ändernder Teile erhöhen.

¤

External Profile Mode

On-board RAM

Alle Positioniersteuerkarten verfügen über On-Board Speicher auf welche in diesem Dokument als On-Board RAM verwiesen wird. Je nach Modell verfügen die Karten entweder über 256k x 8 (700, 800), 512k x 8 (800-B, 900, 1800, 1850, 1900, 1950) Single Port RAM oder 32k x 16 oder 16k x 16 (1920, 1820) Dual Port RAM.

Dieser Speicher dient zwei Zwecken, einmal kann es als Trace Buffer verwendet werden, der andere Zweck ist es für die Speicherung und Ausführung komplexer Bewegungsprofile zu verwenden. Die Verwendung des Speichers für den Trace Buffer erlaubt die real-time Erfassung bestimmter Daten durch den Motion Prozessor und die automatische Speicherung im On-Board RAM. 4 verschiedene Variablen können aus einer Gesamtzahl von 28 ausgewählt werden die gleichzeitig erfasst werden sollen. Die Daten können zu jeder Update-Periode ermittelt werden. Längere Zeitspannen lassen sich einstellen (z.B. nur jede 10. Update-Periode).

Die Länge des Trace Buffers ist variabel und programmierbar. Entweder kann der gesamte Speicher (POSYS® 1800/1900) verwendet werden, oder der gesamte Speicher abzgl. der ersten 512 Worte (POSYS® 700/800/800-B/900).

Falls der Speicher für die Speicherung von Programminformationen und Ausführung benötigt wird, kann mit der gleichen Einschränkung für die POSYS® 700/800/800-B und 900 der gesamte Speicherbereich eingesetzt werden, oder es kann für beide Modi, Trace Buffers und Programm Buffers, verwendet werden. Die unterschiedlichen Buffer werden mit IDs identifiziert, der Trace Buffer muss immer mit ID=0 eingestellt sein.

Die POSYS® 1800/1900 sind optional auch mit Dual Port RAM erhältlich. Nur die Versionen 182x und 192x sind standardmäßig mit 32k x 16 bzw. 16k x 16 Dual Port RAM bestückt. Für alle Versionen der 1800 und 1900 Serie gilt, dass andere Speichergrößen und -arten möglich sind. Anstatt Dual Port RAM kann Single Port RAM und umgekehrt zur

Verfügung gestellt werden. Für Details kontaktieren Sie bitte POSYS Motion Control GmbH & Co.KG.

Relevante On-Board RAM-Funktionen

Funktionen, die relevant für die Verwendung vom On-Board RAM sind:

Profile Mode

- GetBufferFunction
- GetBufferLength
- GetBufferReadIndex
- GetBufferStart
- GetBufferWriteIndex
- GetStopMode
- ReadBuffer
- SetBufferFunction
- SetBufferLength
- SetBufferReadIndex
- SetBufferStart
- SetBufferWriteIndex
- SetStopMode
- WriteBuffer

Die Funktion SetProfileMode(axis, mode) wurde um den Modus "On-Board RAM Mode = 4" erweitert.

Alle die obigen Funktionen und die benötigten Parameter werden ausführlich im Programmier-Handbuch beschrieben.

Trace Mode

- GetTraceCount
- GetTraceMode
- GetTracePeriod
- GetTraceStart
- GetTraceStatus
- GetTraceStop
- GetTraceVariable
- SetTraceMode
- SetTracePeriod
- SetTraceStart
- SetTraceStop
- SetTraceVariable

Diese Funktionen gelten nur für Trace-Funktionen Setup und Ausführung. Eine detaillierte Beschreibung kann im Programmier-Handbuch gefunden werden.

Verwendung von On-Board RAM für External Profile Mode

Der On-Board RAM kann für die Speicherung komplexer Bewegungssequenzen verwendet werden. Die Bewegungsdaten die vom Host kalkuliert wurden, können in den Speicher geladen und dann vom Motion Prozessor ohne weitere Hostintervention ausgeführt werden. Bis zu 4 Achsen gleichzeitig können kontinuierlich in einem hoch-komplexen Pfad bewegt werden.

- Position
- Velocity
- Acceleration
- Jerk (S-curve)
- SegmentTime

Alle Daten sind im 32-Bit Format

- Positionsdaten sind immer Absolut und repräsentieren die Zielposition am Ende des Segments.
- Geschwindigkeit ist in Quadraturzählungen/Update-Periode (Servozyklus)
- Beschleunigung (negativ und positiv) ist in Quadraturzählungen/Update-Periode^2
- Jerk ist in Quadraturzählungen/Update-Periode^3
- SegmentTime ist ein Vielfaches der Update-Periode (Servozyklus)

Wenn alle 4 Achsen aktiviert sind und die Update-Periode ist auf den Standardwert von 408 Mikrosekunden eingestellt, dann ist die schnellste verfügbare Update-Zeit für die Daten im Buffer die durch SegmentTime dargestellt werden ein Multiplikator von 1. Wenn ein SegmentTime Multiplikator von 0 festgestellt wird, wird der External Profile Mode für diese Achse beendet. Die anderen Achsen werden fortfahren mit der Ausführung ihrer Bewegungsprofile. Bewegungsfehler (Schleppabstandsfehler) und ein abrupter Stopp können ebenso ein Ende der Ausführung von Programmen im On-Board RAM herbeiführen.

Die Länge des SegmentTimes hängt vom Profiltyp und von der Applikation ab. Je kürzer die Zeit desto mehr Speicher wird benötigt. Lineare Bewegungsprofile können wesentlich längere SegmentTimes verwenden als zikulare Bewegungsprofile. SegmentTimes für zirkulare Bewegungsprofile benötigen aber in der Regel nie weniger als 2 Millisekunden.

Was bedacht werden muß

Wenn ein Bewegungsprofil im External Profile Mode ausgeführt wird, wird es normalerweise an das Ende des Buffers gelangen, dass zugeteilte Daten enthält. Falls es kein SegmentTimewert mit 0 findet, wird es an die Startposition zurückspringen und mit der Ausführung fortfahren. Dies kann eine potenziell gefährliche Situation darstellen. Falls die Endposition erheblich von der Startposition abweicht, könnte dies dazu führen, dass die Achse einen sofortigen Sprung zu dieser Position durchführt. Um den External Profile Mode zu verlassen, muss das letzte Segment einen SegmentTime Multiplikator von 0 für jede involvierte Achse enthalten. Die Achsen werden dann in ihrer Position verharren. Der ausgewählte Profilmodus (SetProfileMode(axis, mode)) ist dann immer noch aktiv (profil=4) und muss in den gewünschten Modus gesetzt werden für weitere Bewegungsausführungen.

Verfügbarer Speicher (RAM) zu klein für Programmausführung

Die Programmlänge überschreitet den zur Verfügung stehenden Speicher - dies stellt kein Problem dar. Teilen Sie die Bewegungssequenz in kleinere Sequenzen auf. Laden Sie diese Segmente und senden Sie zusätzliche Segmente zum On-Board RAM sobald der Speicher zur Verfügung steht. Der RAM wird in Bereiche aufgeteilt die einmal die aktuelle Sequenz ausführt und einmal die nachfolgenden Daten für die darauffolgende Ausführung bereithält. Die Funktionen GetBufferReadIndex und GetBufferWriteIndex sind Mechanismen um die momentane Leseposition und die neue Schreibposition für neue Daten zu bestimmen. Sobald die Abarbeitung der Anfangsdaten beendet ist, springt der Lesezeiger automatisch zur Startposition und fährt fort mit der Ausführung des Programmes.

Angenommen die folgenden Bedingungen liegen vor: die Gesamtanzahl Programmpositionen die im RAM benötigt werden betragen 26,350, aber nur 10,000 stehen zur Verfügung. Dann fahren Sie folgendermaßen fort:

- Laden Sie die Daten in die Position 0 bis 10,000
- Starten Sie den External Profile Mode
- Setzen Sie einen Unterbrechungspunkt, dass zu einem Ereignis passt sobald der BufferReadIndex die Position 5,000 im RAM passiert, oder prüfen Sie BufferReadIndex >= 5,000
- Beginnen Sie damit neue Daten in den RAM zu laden und beginnen Sie mit der Position BufferWriteIndex = 0 bis zu RAM Position = 4,999
- Prüfen Sie den BufferReadIndex bis er <= 10,000 ist
- Beginnen Sie damit neue Daten in den Speicher zu laden und beginnen Sie damit an der Position BufferWriteIndex = 5,000 und fahren Sie damit fort bis RAM Position = 10,000

Fahren Sie mit dieser Methode fort bis alle Daten geladen und ausgeführt worden sind. Stellen Sie sicher ein Segment anzuhängen mit einem SegmentTime Multiplikator = 0 um den External Profile Mode zu verlassen nach Beendigung der Sequenz.

¤

Das SetStop Kommando

Das SetStop Kommando

Normalerweise würde jedes der vorangegangenen Bahnprofile die entsprechenden Bahnen innerhalb der gesetzten Parameter ausführen bis die Bedingungen erfüllt wurden. Z.B. für ein Punkt-zu-Punkt Profil bedeutet dies, dass das Profil eine Achse solange in Richtung seiner Endposition versucht zu bewegen bis diese erreicht wurde und die Achse wieder gen "0" abbremst.

In manchen Fällen ist es aber notwendig die Achse manuell zu einem Halt zu bringen, z.B. aus Sicherheitsgründen. Dies kann mit zweierlei Methoden erreicht werden: abrupter Stop oder Stop mit programmierter Rampe. Um einen Stop auszuführen, verwendet man den Befehl SetStopMode. Um die gegenwärtige Einstellung für einen Stop abzufragen verwendet man das Kommando GetStopMode.
AbruptStop stoppt die entsprechende Achse ohne Rampe was einem Notfallstop gleichkommt. Dabei wird die Geschwindigkeit der Achse sofort auf "0" gesetzt.

SmoothStop dagegen stoppt die entsprechende Achse mit dem jeweilig eingestellten Wert für die negative Beschleunigung. Ist dieser Wert = 0 wird dann der Wert für die positive Beschleunigung verwendet.

VORSICHT
Abrupt Stop muss mit Vorsicht verwendet werden. Plötzliches Abbremsen aus einer hohen Geschwindigkeit kann die Mechanik beschädigen oder zu Verletzungen führen.

AbruptStop funktioniert in allen Profilen. SmoothStop funktioniert in allen Profilen, außer Electronic-Gear.

¤

Motor Mode

Motor Mode

Alle POSYS® unterstützen einen programmierbaren Motormodus, welches innerhalb des Profilgenerators aktiviert bzw. deaktiviert werden kann und im Servomodus kann die POSYS® entweder im Open Loop Modus oder im Closed Loop Modus gesetzt werden (jede Achse individuell).

Der Befehl SetMotorMode setzt den Modus und der Befehl GetMotorMode liefert den gegenwärtigen Modus zurück.
Falls der MotorModus auf AN geschaltet ist, dann ist der Profilgenerator eingeschaltet. Falls der MotorModus auf AUS geschaltet ist, ist der Profilgenerator ausgeschaltet.

Außerdem, für die Achsen der POSYS® im Servomodus gilt, wenn der Motormodus auf AUS gesetzt ist, dann arbeitet die POSYS® im Open Loop Modus, welches bedeutet dass der Servofilter ausgeschaltet ist und die Spannungsausgabe der von der POSYS® erzeugt wird, vom Host gesetzt wird indem er den Befehl SetMotorCommand benutzt. Wenn der Motormodus eingeschaltet ist dann wird die Spannungsausgabe durch die POSYS® und der programmierten Parameter bestimmt.

Die meist gebräuchliche Verwendung des Motormodus ausser der Standardeinstellung "ON", ist nach einem Schleppabstandsfehler. In diesem Fall (und falls AutoStop aktiviert is) setzt die POSYS® den MotorModus automatisch auf AUS, dabei es in einen sicheren Status setzend bei der keine weitere Bewegung möglich ist bis der Host die Parameter und Modi explizit wieder herstellt und in den "ON"-Modus schaltet.

Für die Achsen der POSYS® im Servomodus gilt auch, dass es sinnvoll sein kann den MotorModus auf "AUS" zu schalten zum Zwecke der Verstärkerkalibrierung.

¤

Zykluszeit

Zykluszeit setzen

POSYS® 7xx/8xx/8xx-B/9xx

Die POSYS® kalkuliert jede Bahn und Servoinformation in einem festen Zeitabstand. Dieser Abstand wird die Zykluszeit (cycle time) genannt.

Für jede aktivierte Achse der POSYS® gibt es eine minimale Zykluszeit, damit die POSYS® alle anfallenden Kalkulationen erledigen kann. Die folgende Tabelle zeigt die Unterschiede:

POSYS®	Zykluszeit pro aktivierte Achse
POSYS® 70x, 80x, 81x, 90x, 91x	100 µsec
POSYS® 75x, 85x, 95x	n.v. (anstatt: Profile Calculation Timing Range (keine PID-Regelung möglich) = 100 µsec POSYS® 75x, 85x, 95x; 150 µsec for POSYS® 84x, 86x, 94x, 96x
POSYS® 82x, 83x, 87x, 88x, 92x, 93x, 97x, 98x	150 µsec

Um die minimale Zykluszeit für eine gegebene Anzahl von aktivierten Achsen berechnen zu können, multiplizieren Sie die Anzahl Achsen mit der minimalen Zykluszeit. z.B. für eine POSYS® 904 mit 4 aktivierten Achsen wäre die minimale Zykluszeit 4x100 = 400µsec, welches eine Frequenz von 2,5 kHz entspricht.

Die Zykluszeit bestimmt die Servo-Loop-Updaterate anhand der die Parameter pro aktivierte Achse neu kalkuliert und angeglichen werden. Jedoch bestimmt es nicht die Kommutierungsrate für bürstenlose Servomotoren.

Jede aktivierte Achse erhält sein „Zeitfenster“ egal ob es in Bewegung ist oder nicht, egal ob mit dem Kommando SetMotorMode im EIN oder AUS Zustand ist. Falls die Zykluszeit kritisch ist, ist es möglich die Zykluszeit zu verkürzen in dem unbenutzte Achsen deaktiviert werden (SetAxisMode) und in dem die Servo-Loop-Updaterate mit dem Kommando SetSampleTime neu gesetzt wird.

z.B., bei Verwendung einer 4-Achsen POSYS 704, 804, 904 sind standardmässig 4 Achsen aktiviert, aber falls nur 3 benötigt werden wird mit dem Kommando SetMotorMode eine Achse deaktiviert und mit dem Kommando SetSampleTime eine neue Updatrate von 300 µsec programmiert. Dies verbessert die Frequenz von 2.5 kHz auf 3.333 kHz. SetSampleTime kann auch verwendet werden um die Zykluszeit zu verlängern, falls dies notwendig sein sollte.

POSYS® 18xx/19xx

Die minimale Zykluszeit für eine POSYS® 18xx/19xx ist 64 µsec, abhängig von der Anzahl Achsen und Motorvarianten die möglicherweise angeschlossen sind. Mit dieser Einstellung ist die bestmögliche Frequenz 9.76 kHz. Diese Frequenz kann in 64/68,3/76,8/102,4 µsec Schritten erhöht werden.

Für jede unterschiedliche Anzahl aktivierter Achsen der POSYS® 18xx/19xx gibt es unterschiedlich große "Zeitfenster". Die folgende Tabelle listet diese auf:

Für POSYS 1800/1900 mit Motion Control Chipsatz Ver. <=1.5

POSYS®	Zykluszeit pro aktivierte Achse	Min. Zykluszeit mit Tracefunktion	Zeit pro Achse	Max. Zyklusfrequenz
1 Achse	51,2 µs	102,4 µs	51,2 µs	19,53 kHz
2 Achsen	153,6 µs	153,6 µs	76,8 µs	6,51 kHz
3 Achsen	204,8 µs	204,8 µs	68,3 µs	4,88 kHz
4 Achsen	256 µs	256 µs	64 µs	3,91 kHz

Für POSYS 1800/1900 mit Motion Control Chipsatz Ver. >2.x

POSYS®	Zykluszeit pro aktivierte Achse	Min. Zykluszeit mit Tracefunktion	Zeit pro Achse	Max. Zyklusfrequenz
1 Achse	102,4 µs	102,4 µs	102,4 µs	9,76 kHz
2 Achsen	153,6 µs	153,6 µs	76,8 µs	6,51 kHz
3 Achsen	204,8 µs	204,8 µs	68,3 µs	4,88 kHz
4 Achsen	256 µs	256 µs	64 µs	3,91 kHz

Die Zykluszeit bestimmt die Servo-Loop-Updaterate anhand der die Parameter pro aktivierte Achse neu kalkuliert und angeglichen werden. Jedoch bestimmt es nicht die Kommutierungsrate für bürstenlose Servomotoren.

Jede aktivierte Achse erhält sein „Zeitfenster“ egal ob es in Bewegung ist oder nicht, egal ob mit dem Kommando SetMotorMode im EIN oder AUS Zustand ist. Falls die Zykluszeit kritisch ist, ist es möglich die Zykluszeit zu verkürzen in dem unbenutzte Achsen deaktiviert werden (SetAxisMode) und in dem die Servo-Loop-Updaterate mit dem Kommando SetSampleTime neu gesetzt wird.

z.B., bei Verwendung der POSYS® ist es möglich mit 4 Achsen zu operieren, aber falls in einer spezifischen Anwendung nur 3 Achsen verwendet werden, kann die vierte Achse mit dem Kommando SetAxisMode deaktiviert und eine neue Zykluszeit von 204,8 µs gesetzt werden. Dies verbessert die Frequenz von 3,9 kHz auf 4,88 kHz.

SetSampleTime kann auch verwendet werden um die Zykluszeit zu verlängern, falls dies notwendig sein sollte.

Es liegt in der Verantwortung des Anwenders dafür Sorge zu tragen, dass die Zykluszeit gleich oder größer als die spezifizierten Minimalwerte, wie in der Tabelle voraus beschrieben, sind. Werte die kleiner als die kleinstmöglichen Einstellungen sind, werden automatisch auf die Minimalwerte zurückgesetzt.

¤

Parameter Update

Parameter Buffering

Parameter Buffering

Verschiedene Parameter müssen für die POSYS® angegeben werden, damit eine Achse richtig kontrolliert wird. In einigen Fällen kann es wünschenswert sein, dass einige Parameter zur exakt gleichen Zeit aktiv/aktuell werden um exakt synchronisierte Bewegungsabläufe zu gewährleisten.

Um diese Arten von Profilparametern und einige andere Arten von Parametern wie Servoparameter zu unterstützen (POSYS® Achsen in Servomodus) werden sie in Buffer geladen. Diese zwischengespeicherten Befehle werden in einen Bereich der POSYS® geladen, der die eigentliche POSYS®-Verhaltensweise nicht beeinflusst, bis ein besonderes Ereignis, bekannt als ein Update, eintritt. Ein Update veranlasst die zwischengespeicherten Register in die aktiven Register zu laden, die POSYS® dazu veranlassend die neuen Parameter zu verwenden.

Zum Beispiel: die folgende Kommandosequenz lädt ein Profilmodus, Positionswert, Geschwindigkeit, und Endposition, aber sie werden nicht aktiv bis ein Updatekommando erteilt wird:

Kommando	Aktion
SetProfileMode(Axis1, trapezoidal)	setze Profilmodus trapezförmig für Achse 1
SetPosition(Axis1,12345)	lade Endposition für Achse 1
SetVelocity(Axis1, 223344)	lade Geschwindigkeit für Achse 1
SetAcceleration(Axis1,1000)	lade Beschleunigung für Achse 1

Nachdem diese Sequenz abgeschlossen wurde, werden die zwischengespeicherten Register für diese Parameter (einschließlich des Profilmodus selbst) in die Register der POSYS® geladen, aber das Bahnprofil läuft noch nach den vorhergehenden Werten ab. Nur wenn ein Update-Kommando ausgeführt wird, wird der Profilmodus eigentlich zu trapezförmig und die angegebenen Parameter in den Trajektoriegenerator geladen die die POSYS® sofort veranlassen, das angegebene Programm zu starten.

Updates

Es gibt drei unterschiedliche Methoden für ein Update. Sie werden nachfolgend aufgelistet:

- Update Kommando - Die einfachste Art ist, ein Update-Kommando zu geben. Dies veranlasst die Parameter für die programmierte Achse sofort zu aktualisieren.

- MultiUpdate Kommando - Der multiple Achsenupdate, dass bei der Verwendung des MuItiUpdate-Befehles angegeben wird, veranlasst, dass mehrere Achsen gleichzeitig aktualisiert werden. Dies kann nützlich sein, wenn interpolierende Achsen synchronisiert werden sollen. Dieses Kommando nimmt ein 1-Wort Argument aus einer Bitmaske, welches aus einem Bit besteht und jeder Achse zugewiesen ist. Dieser Befehl hat bei Ausführung die gleiche Wirkung wie das Update-Kommando, welches an jede Achse einzeln geschickt wird.

- Breakpoints - Es gibt eine sehr nützliche Funktion die von der POSYS® unterstützt wird, die programmiert werden kann, ein Update-Befehl automatisch zu generieren - wenn ein vorprogrammierter Zustand wahr wird. Diese Möglichkeit wird Unterbrechungspunkt genannt, und es ist nützlich für Operationen wie z.B.: "automatisch die Geschwindigkeit verändern wenn eine besondere Position erreicht wird", oder "die Achse abrupt stoppen wenn ein bestimmtes externes Signal aktiv wird".

Gleich welche Update-Methode benutzt wird, zu dem Zeitpunkt zu dem der Update passiert, werden alle zwischengespeicherten Register zu den aktiven Registern kopiert. Umgekehrt, bevor das Update passiert, haben zwischengespeicherte Register oder die Ausführung zwischengespeicherter Befehle so lange keinen Effekt auf das laufende System bis das Update-Kommando tatsächlich ausgeführt wurde.
Im Gegensatz zur Profilerstellung sind die meisten Parameterkommandos nicht gebuffert und einige sind gebuffert. Die folgende Liste zeigt alle gebufferten Kommandos.

Zwischengespeicherte Befehle

Trajectory	Servo & Error Tracking	Verschiedenes
SetProfileMode	ClearPositionError	SetMotorCommand

Trajectory	Servo & Error Tracking	Verschiedenes
SetAcceleration	SetIntegrationLimit	
SetJerk	SetKaff	
SetVelocity	SetKd	
SetPosition	SetKi	
SetDeceleration	SetKp	
SetGearRatio	SetKvff	
	SetKaff	

¤

Breakpoints

Breakpoints

Breakpoints

Breakpoints sind eine günstige Methode, Ereignisse auf einige spezifische Zustände hin zu programmieren. Je nach den Argumenten der Breakpointanweisung kann ein Breakpoint ein Update verursachen; ein abrupter Stopp nach einem Update, ein SmoothStop nach einem Update, ein Motor OFF gefolgt von einem Update (mehr zu dieser Funktion in einem späteren Abschnitt); oder keine Aktion was auch immer.

Jede Achse kann für die Verwendung von 2 Breakpoints programmiert werden. So können zwei vollkommen gesonderte Zustände überwacht und daraufhin auf Ereignisse ausgelöst werden. Diese zwei Breakpoints werden Breakpoint 1 und Breakpoint 2 genannt.

¤

Einen Breakpoint definieren; Übersicht

Einen Breakpoint definieren, Übersicht

Jeder Breakpoint hat fünf Komponenten: die Breakpointachse, die Sourceachse für das Triggerereignis, das Ereignis selbst, die auszuführende Aktion und den Vergleichswert.

Die Breakpointachse ist die Achse, auf der die angegebene Aktion ausgeführt werden soll.

Die Sourceachse ist die Achse, auf der das Triggerereignis stattfindet. Es kann die Gleiche oder eine andere als die Breakpointachse sein. Irgendeine Anzahl von Breakpoints kann die gleiche Achse wie die Sourceachse benutzen.

Der Trigger ist das Ereignis, der den Breakpoint auslöst.

Die Aktion ist der Aufgaben-Ablauf der von der POSYS® ausgeführt wird, wenn der Breakpoint getriggert wird.

Nachdem ein Breakpoint getriggert wird, wird die Aktion auf der Breakpointachse ausgeführt. Der Vergleichswert wird benutzt, zusammen mit der Aktion, um das Breakpointereignis zu definieren.
Allzusammen erlauben diese Parameter große Flexibilität um Breakpointzustände zu definieren. Durch das Verknüpfen dieser Komponenten, kann fast jedes Ereignis auf jeder Achse einen Breakpoint verursachen.

Der Befehl SetBreakPoint wird benutzt um die Breakpointachse, den Trigger, die Sourceachse und die Aktion zu definieren. Um diese Werte zurück zu lesen wird der Befehl GetBreakpoint benutzt. Der Befehl SetBreakpointValue wird benutzt den Vergleichswert zu setzen. Der Befehl GetBreakpointValue liefert den Vergleichswert zurück. Die Breakpointnummer (1 oder 2) muss für jeden dieser Befehle angegeben werden.

Der SetBreakpointValue-Befehl sollte immer vor dem SetBreakpoint-Befehl gesendet werden um einen besonderen Breakpoint einzurichten.

¤

Breakpoint Trigger

Breakpoint Trigger

Die POSYS® unterstützt die folgenden Breakpoint Trigger Zustände:

Trigger Kondition	Level or Threshold	Beschreibung
Positive Soll-Position	Threshold	Ist gegeben wenn die gegenwärtige Sollposition gleich ist oder größer als der programmierte Vergleichswert

Trigger Kondition	Level or Threshold	Beschreibung
Negative Soll-Position	Threshold	Ist gegeben wenn die Sollposition gleich ist oder kleiner als der programmierte Vergleichswert
Positive Ist-Position	Threshold	Ist gegeben wenn die Istposition gleich ist oder größer als der programmierte Vergleichswert
Negative Ist-Position	Threshold	Ist gegeben wenn die Istposition gleich ist oder kleienr als der programmierte Vergleichswert
Soll-Position überschritten	Threshold	Ist gegeben wenn die Istposition sich kreuzt (ist gleich zu) mit dem programmierten Vergleichswert
Ist-Position überschritten	Threshold	Ist gegeben wenn die Istposition sich kreuzt (ist gleich) der programmierte Vergleichswert
Zeit	Threshold	Ist gegeben wenn die gegenwärtige POSYS®-Zeit (in Zyklenzah seit Einschalten) ist gleich dem programmierten Vergleichswert
Event Status	Level	Ist gegeben wenn der EventStatusRegister der Bitmaske und de high/low Pattern der programmierte Vergleichswert ist
Activity Status	Level	Ist gegeben wenn der ActivityStatusRegister der Bitmaske gleicl und der high/low Pattern der programmierte Vergleichswert ist
Signal Status	Level	Ist gegeben wenn der SignalStatusRegister der Bitmaske gleicht und der high/low Pattern der programmierte Vergleichswert ist
Keine	-	Deaktivieren Sie einen früher gesetzten Breakpoint

Wenn "Keine" für den Breakpointtrigger selektiert wird, bedeutet dies dann tatsächlich, dass jener Breakpoint inaktiv ist. Nur einer der obigen Trigger kann zur gleichen Zeit selektiert werden.

Threshold-triggered Breakpoints

Threshold triggered Breakpoints verwenden den mit dem Kommando SetBreakpointValue ermittelten 32-bit Schwellwert als Vergleichswert. Wenn der Vergleich wahr ist, wird der Breakpoint getriggert. Zum Beispiel wenn es gewünscht wird dass getriggert wird, wenn die Sollposition gleich oder größer 1000000 ist, dann ist der Vergleichswert der mit dem Kommando SetComparisonValue geladen wird 1000000 und der ausgewählte Trigger wäre PositiveCommandedPosition.

Level-triggered Breakpoints

Der Host liefert zwei 16-Bit Daten-Wörter einen level-getriggerten Breakpoint zu setzen: eine Triggermaske und eine Zustandsmaske (high oder low). Diese Werte werden gesetzt mit dem Kommando SetBreakpointValue. Die oberen Bits des Datenwortes sind der Wert für die Triggermaske und die unteren Bits des Datenwortes sind der Wert für die Zustandsmaske.

Die Triggermaske bestimmt, welche Bits des ausgewählten Zustandsregisters für den Breakpoint freigegeben werden. Eine ''1'' in jeder beliebigen Position der Triggermaske gibt das entsprechende Zustandsregisterbit frei einen Breakpoint zu triggern, eine ''0'' in der Triggermaske sperrt das entsprechende Zustandsregisterbit. Wenn mehr als ein Bit selektiert wird, wird der Breakpoint getriggert dessen spezifizierter Status erreicht wird.

Die Zustandsmaske bestimmt, welcher Status des entsprechenden Statusbits einen Breakpoint erzeugt. Jeder Statusbit, das in dem gleichen Zustand (z.B. 1 oder 0) wie das entsprechende Zustandsbit ist, ist geeignet, einen Breakpoint auszulösen (vorausgesetzt, dass es von der Triggermaske selektiert worden ist).

Zum Beispiel wenn der Statusregisterbreakpoint selektiert worden ist, und die Triggermaske den Wert 0402h und die Zustandsmaske den Wert 0002h enthält wird der Breakpoint dann getriggert wenn Bit 1 (der 'at max. velocity' Indikator) den Wert 1 annimmt, oder Bit 10 (der 'in Bewegung' Indikator) den Wert übernimmt.

¤

Breakpoint Aktionen

Breakpoint Aktionen

Sobald ein Breakpoint getriggert worden ist, kann die POSYS® programmiert werden, eine der Folgeanweisungen auszuführen:

Aktion	Ausgeführtes Kommando
Keine	Keine Kommandos werden ausgeführt
Update	Update Achse
Abrupter Stopp	SetStopMode axis, AbruptStop Update axis
SmoothStop	SetStopMode axis, SmoothStop Update axis
MotorOff	SetMotorMode axis, Off Update axis

Ungeachtet der Aktion des Hosts, ist einmal einem Breakpointzustand entsprochen worden, das dem Breakpoint entsprechende Ereignis-Zustands-Bit wird gesetzt und der Breakpoint wird deaktiviert.

¤

Breakpoint Beispiele

Hier sind ein paar Beispiele die veranschaulichen sollen wie Breakpoints benutzt werden können.

Beispiel #1: Der Host möchte, dass Achse 1 die Geschwindigkeit verändert, wenn die Enkoderposition einen besonderen Wert erreicht. Breakpoint #1 soll benutzt werden.

Die folgende Kommandosequenz erreicht dies:

Kommando	Beschreibung
SetPosition(Axis1,123456)	Lade Endposition
SetVelocity(Axis1, 55555)	Lade Geschwindigkeit
SetAcceleration(Axis1, 500)	Lade Beschleunigung
SetDeceleration(Axis1,1000)	Lade negative Beschleunigung
Update(Axis1)	Starte
SetVelocity(Axis1,111111)	Lade neue Geschwindigkeit von 111.111 aber noch nicht das Update schicken
SetBreakpointValue(Axis1,1,100000)	Lade 100000 in das Vergleichsregister für Breakpointregister 1
SetBreakpoint(Axis1, Breakpoint1, Axis1, Update, PositiveActualPosition)	spezifiziere eine positive aktuelle Positionsbreakpoint auf Achse 1 welches in einem Update endet wenn Breakpoint1 eintritt

Diese Sequenz startet eine Bewegung und lädt einen Breakpoint, nachdem die erste Bewegung gestartet wurde. Der Breakpoint, der definiert wurde, wird die Geschwindigkeit updaten zu 111111, wenn die Istposition einen Wert von 100000 erreicht. Deswegen wird sich die Achse an Position 100000 von einer Geschwindigkeit von 55555 bis 111111 mit dem Beschleunigungswert von 500 beschleunigen.

Anmerkung: zwischengespeicherte Register, die nicht nochmals gesandt werden, verbleiben in den zwischengespeicherten Registern. Zum Beispiel wenn der Breakpoint ein Update ausführt, sind die Werte für Position, Beschleunigung und Verlangsamung unverändert und werden deswegen zu den aktiven Registern ohne Modifikation kopiert.

Beispiel #2: Der Host möchte, dass Achse 1 einen Notstopp ausführt, wann auch immer das AxisIn-Signal für Achse 3 auf „high“ geht. Außerdem die Beschleunigung der Achse 1 soll verändert werden, wann immer eine besondere Istposition auf Achse 4 erreicht wird.

Die folgende Kommandosequenz erreicht dies:

Kommando	Beschreibung
SetPosition(Axis1,123456)	Lade Endposition
SetVelocity(Axis1, 55555)	Lade Geschwindigkeit
SetAcceleration(Axis1, 500)	Lade Beschleunigung
SetDeceleration(Axis1, 1000)	Lade negative Beschleunigung
Update(Axis1)	Start Bewegung
SetBreakpointValue(Axis1, 1, 0x400040)	Lade Maske und Zustandswort von 0x40, 0x40 (bit 6 muß high sein) für Breakpoint 1

Kommando	Beschreibung
SetBreakpoint(Axis1, Breakpoint1, Axis3, AbruptStop, SignalStatus)	Spezifiziere einen Breakpoint um den SignalStatusRegister von Achse zu überwachen und zu triggern falls Bit 6 auf high geht für Breakpoint
SetAcceleration(Axis1, 5000)	Lade eine neue Beschleunigung von 5000 aber schicke noch kein Update
SetBreakpointValue(Axis1, 2,100,000)	Lade 100.000 in das Vergleichsregister für Breakpoint 2
SetBreakpoint(Axis1, Breakpoint2, Axis4, Update, PositiveCommandedPosition)	spezifiziere einen Breakpoint für die positive Sollposition auf Achse 4 welches in einem Update resultiert wenn Zustand gegeben für Breakpoint 2

Diese Sequenz ist dem vorherigen ähnlich, ausser dass ein zusätzlicher Breakpoint definiert worden ist, der den abrupten Stopp veranlasst. Beide Breakpoints wurden definiert, nachdem die Bewegung begonnen wurde, obwohl dies nicht unbedingt notwendig ist, je danach, wann man erwartet, dass der Breakpoint vorkommt. Generell sollten Breakpoints nach der primären Bewegungsinitiierung eingerichtet werden, weil es nur einen Satz gebufferter Register gibt und es damit unmöglich ist primäre Bewegungsparameter als auch (Position, Geschwindigkeit, etc.) "Breakpoint"-Parameter zu laden (jene Parameter die aktuell werden sollen, nachdem der Breakpoint aktiviert wurde) bevor die eigentliche Bewegung aktualisiert wurde.

¤

Der Servo Loop

Überblick

Überblick

Für die Achsen der POSYS® die im Servomodus betrieben werden, wird die Regelschleife als Teil der Grundmethode benutzt, den Drehmomentsollwert zu bestimmen. Der Grundgedanke der Regelschleife ist die Istposition so nahe wie möglich mit der Sollposition gleichzusetzen und entsprechende Angleichungen durch Anpassung des Drehmomentsollwertsignals dahingehend vorzunehmen, dass entsprechend den Parametervorgaben die Soll- und Istposition möglichst identisch sind.

Um dies zu bewerkstelligen wird der Ausgangswert mit der aktuellen Enkoderposition verglichen um den Schleppabstandsfehler zu errechnen, welches wiederum durch die Regelschleife geschickt wird. Das skalierte Ergebnis der Filterkalkulation ist die Drehmomentsollwertvorgabe, welches als impulsbreiten moduliertes Signal mit 10-Bit Auflösung und 20 kHz, 8-Bit Auflösung mit 80 Khz oder als 16-Bit Wert an einen D/A-Konverter geschickt wird um ein ±10V analoges Ausgangssignal zu generieren.

¤

PID Regelalgorithmus

PID Regelalgorithmus

Der mit der POSYS® benutzte Servofilter ist ein PID-Algorithmus mit "velocity und acceleration feedforward" Termini und einem Ausgangsmaßstabsfaktor. Ein Integrationslimit sichert nach oben ab gegen einen akkumulierten Fehler. Ein optionaler "bias" kann zur Filterkalkulation addiert werden, um die letzte Motorausgabeanweisung zu erzeugen. Ein Grenzwert für den Filterausgang sorgt für eine zusätzliche Beschränkung.

Die PID+Vff+Aff Formel, zusammen mit dem Skalierungsfaktor und den BIAS termini, ist wie folgt:

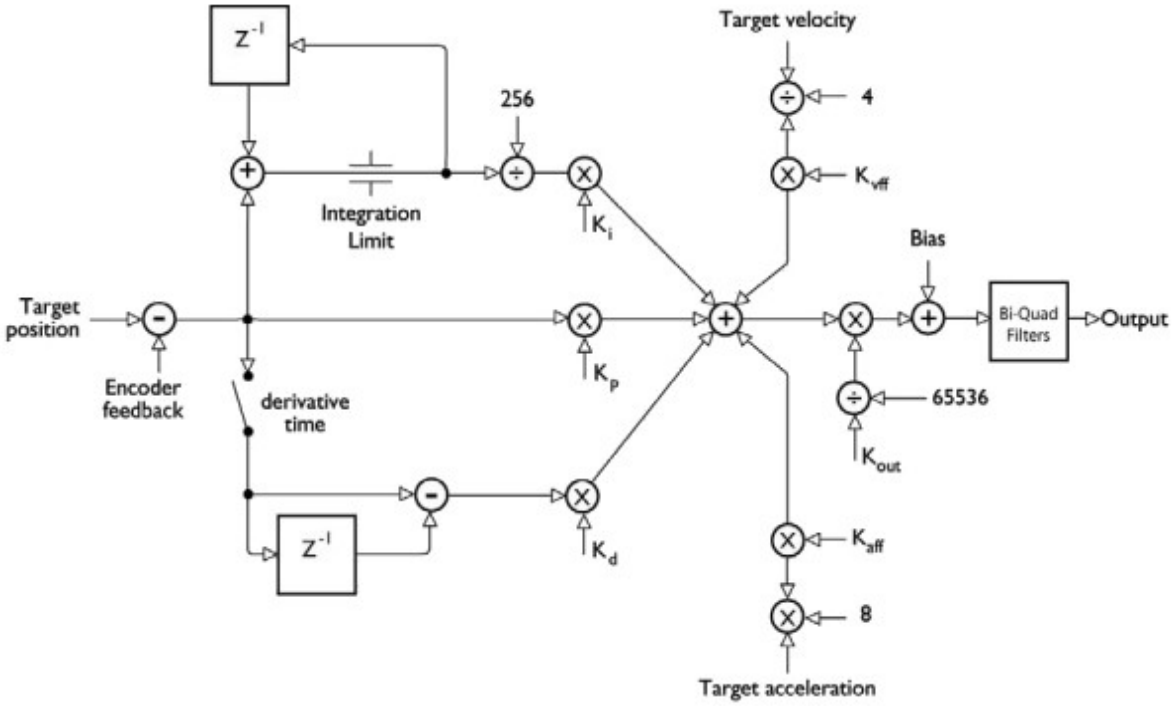
$$Output_n = \left[K_p E_n + K_d (E_n - E_{(n-1)}) + \sum_{j=0}^{n} E_j * Ki/256 + K_{vff}(CmdVel/4) + K_{aff}(CmdAccel/8)\right] * K_{out}/65536 + Bias$$

Wobei E_n	sind die akkumulierten Fehler Termini
K_I	ist der “Integral Gain”
K_d	ist der “Derivative Gain”
K_P	ist der “Proportional Gain”
K_{aff}	ist der “Acceleration feed-forward”
K_{vff}	ist der “Velocity feed-forward”
Bias	ist der „DC motor offset“
K_{out}	ist der Skalierungsfaktor für das Drehmomentsollwertsignal

Alle Filterparameter, die Drehmomentsignalbegrenzung, und der "Motor Bias" sind programmierbar, so dass der Filter durch das Anwenderprogramm fein abgestimmt werden kann. Die einstellbaren Bereiche für die Parameter, Formate und Interpretationen werden in der folgenden Tabelle gezeigt:

Terminus	Name	Bereich
I_{lim}	Integration Limit	unsigned 32 Bit (0 to 2.124.483.647)
K_I	Integral Gain	unsigned 16 Bit (0 to 32.767)
K_d	Derivative Gain	unsigned 16 Bit (0 to 32.767)

Terminus	Name	Bereich
K_p	Proportional Gain	unsigned 16 Bit (0 to 32.767)
K_{aff}	Acceleration feed-forward	unsigned 16 Bit (0 to 32.767)
K_{vff}	Velocity feed-forward	unsigned 16 Bit (0 to 32.767)
K_{out}	Output scale factor	unsigned 16 Bit (0 to 32.767)
Bias	DC motor offset	signed 16 Bit (32.767 to 65.535)
	Motor command limit	unsigned 16 Bit (0 to 32.767)



Die Struktur des digitalen Filters

Motor bias

Wenn eine Achse abhängig von einer externen Kraft in einer Richtung belastet wird (wie eine vertikale abwärts von der Schwerkraft belastete Achse), kann dies der Servofilter durch eine konstante DC-Vorspannung am Filterausgang kompensieren. Der Vorspannungswert wird mit der Verwendung des Kommandos SetMotorBias (axis, bias) gesetzt. Der programmierte Wert kann durch den Befehl GetMotorBias (Achse) gelesen werden.

Outputskalierung

Der Kout Parameter kann dazu verwendet werden die PID-Regelparamter, in Situationen in denen dies notwendig erscheint, herunter zu skalieren. Dies wird dadurch bewerkstelligt indem das Filterergebnis mit Kout/65536 multipliziert wird. Der Effekt ist, dass der Bereich für Kp erweitert wird, welches typischerweise innerhalb eines Bereiches von 1 bis 150 liegt wenn keine Ausgangsskalierung vorgenommen wird. Der Kout-Wert wird mit dem Kommando SetKout gesetzt. Mit GetKout wird der programmierte Wert gelesen.

Output limit

Die Motorausgangsgrenze hält den Filterausgang davon ab, einen Motorausgangsgrenzwert in jeder Richtung zu überschreiten. Wenn der Filter einen Wert, größer als die Grenze, erzeugt, nimmt das Kommando für das Motorausgangssignal den Grenzwert.

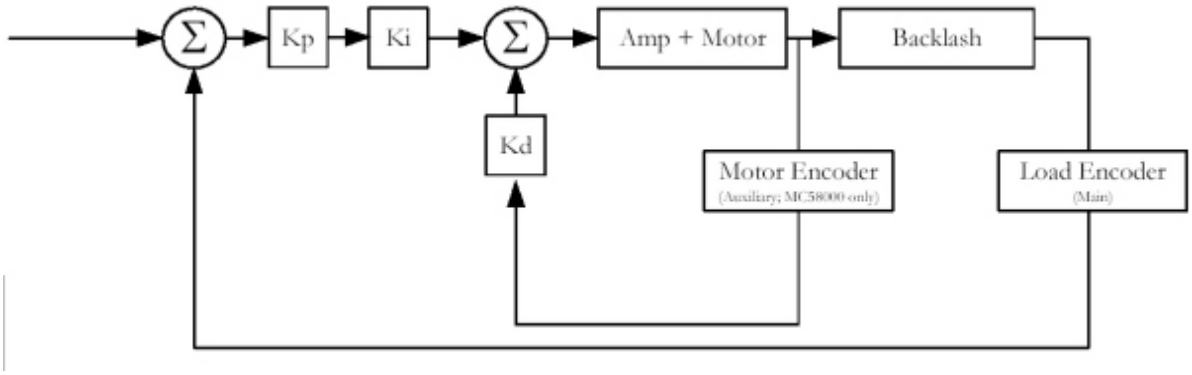
Die Begrenzung des Motorausgangssignals ist nur im Closed-Loop-Modus möglich. Es beeinflusst nicht das Drehmomentsollwertsignal im Open-Loop-Modus.

¤

Dual Encoder Support

Dual-Enkoder Unterstützung

Die 2-4 Achsen Versionen der POSYS® 18xx/19xx Serien unterstützen Dual-Enkoder-Rückmeldung. Dies ist besonders dann von Nutzen wenn die tatsächliche Position z.B. aufgrund von Backlash von der gemeldeten Position abweicht. In dieser Konfiguration ist die Enkoderrückmeldung einer 2. Achse verknüpft mit der zu kontrollierenden Achse als „Derivative Term“. Wenn der Enkodereingang der 2. Achse in dieser Weise verwendet wird, steht sie für einen 2. Motor als Rückmeldeeingang nicht mehr zur Verfügung.



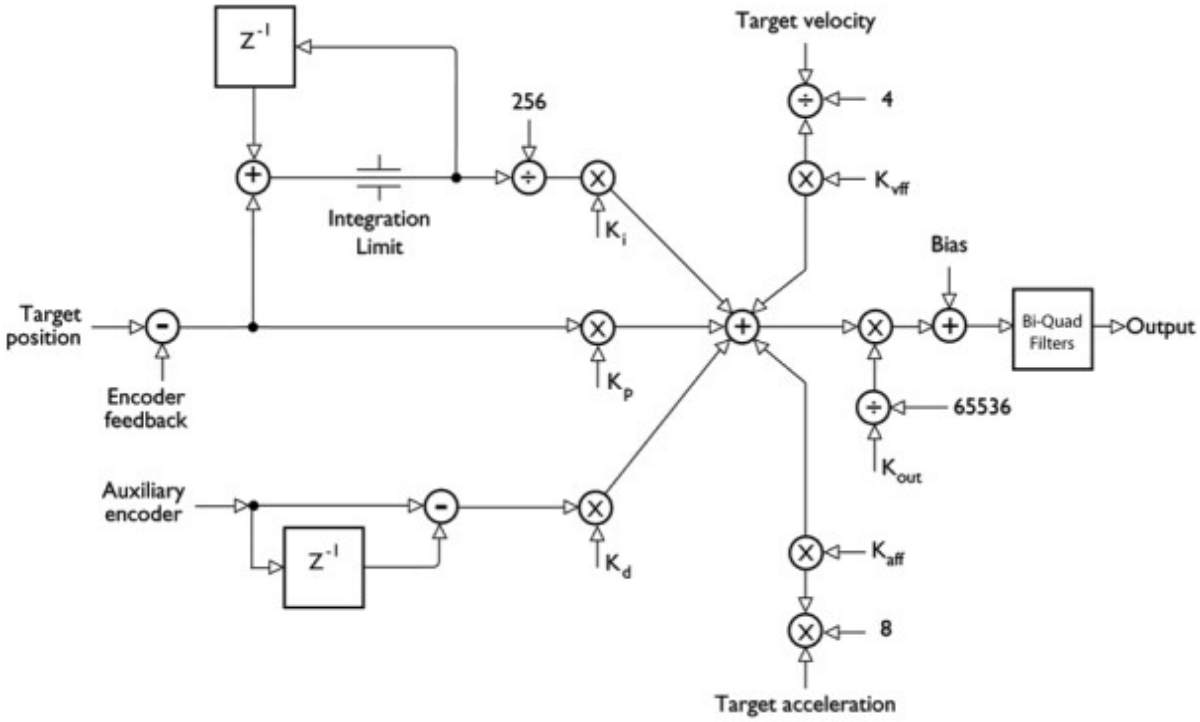
Dual Enkodereingang

Dual Enkoder PID Loop Algorithmus

Die Formel die für den Servofilter Anwendung findet in Verbindung mit dem Dual-Enkoder Algorithmus weicht von der Formel für die normale Verwendung des Servofilters in folgender Weise ab:

$$Output_n=\left[K_P E1_n-K_d(P_n-P_{(n-1)})+\sum_{j=0}^{n} E1_j*Ki/256+K_{vff}(CmdVel/4)+K_{aff}(CmdAccel/8)\right]*K_{out}/65536+Bias$$

Wobei $E1_n$	sind die akkumulierten Fehler vom 1. Enkoder
P_n	Ist die Position des 2. Enkoders
K_I	ist der “Integral Gain”
K_d	ist der “Derivative Gain”
K_P	ist der “Proportional Gain”
K_{aff}	ist der “Acceleration feed-forward”
K_{vff}	ist der “Velocity feed-forward”
Bias	ist der “DC motor offset”
K_{out}	ist der Skalierungsfaktor für das Drehmomentsollwertsignal



POSYS Dual Loop Digitalfilter

Konfiguration von Dual-Enkoder Unterstützung

Das Kommando SetAuxiliaryEncoderSource wird verwendet um die Dual-Enkoder Unterstützung zu aktivieren. Der auxiliaryAxis Parameter bestimmt welcher Achsenenkodereingang verwendet wird um die primäre Achse zu unterstützen. Der mode Parameter aktiviert bzw. deaktiviert den Dual-Enkoder-Modus. Solange der Dual-Enkoder-Modus deaktiviert ist, funktioniert der Servofilter wie unter PID Schleifenalgorithmus beschrieben.

Falls die Anwendung die aktuelle Position des Auxiliary Enkoders bestimmen muss, dann verwenden Sie das Kommando GetActualPosition mit Achsenspezifizierung des Auxiliary Enkoders.

Achtung
Der Auxiliary-Enkoder sollte eine Auflösung aufweisen die mindestens genauso fein oder feiner ist als die des Hauptenkoders. Dies wird helfen potenziell instabile Systemkonditionen zu vemeiden.

¤

BiQuad Ausgangsfilter

Bi-Quad Ausgangsfilter

Ein Bi-Quad ist eine generische digitale Filterstruktur. Mit den geeigneten Koeffizienten kann sie als Low-Pass Filter, High-Pass Filter, Band-Pass Filter, Notch Filter oder kundenspezifischer Filter programmiert werden. Programme wie von Octave (www.octave.org) können verwendet werden um die passenden Koeffizienten zu ermitteln.

Die POSYS® unterstützt bis zu 2 programmierbare Bi-Quad Ausgangsfilter für jede Achse. Diese Filter sind verknüpft; wenn beide aktiviert sind, verwendet Filter1 die Ausgabe von Filter0. Falls Filter0 deaktiviert ist (Standard) dann wird der gesamte Filter umgangen und der Motorausgang passiert ungefiltert.

Die Ausgabe des Filters zum Zeitpunkt n ist folgendermaßen definiert:

$$Y_n = K*(B_0*X_0+B_1*X_{n-1}+B_2*X_{n-2}+A_1*Y_{n-1}+A_2*Y_{n-2})$$

Wobei Y_n	Ist die Ausgabe des Filters zum Zeitpunkt n

X_n	Ist der Eingang zum Filter zum Zeitpunkt n
K	Ist positiver Skalierungswert um Rundungsfehler zu vermeiden
B0	Ist ein programmierbarer Bi-Quad Koeffizient
B1	Ist ein programmierbarer Bi-Quad Koeffizient
B2	Ist ein programmierbarer Bi-Quad Koeffizient
A1	Ist ein programmierbarer Bi-Quad Koeffizient
A2	Ist ein programmierbarer Bi-Quad Koeffizient

Bestimmung der Bi-Quad Koeffizienten

Typischerweise sind Koeffizienten, die in Bi-Quad Filtergleichungen verwendet werden, kleine Fließkommazahlen. Um daher Rundungsfehler zu vermeiden wenn diese Zahlen als 16-Bit Werte gespeichert werden, wird der K-Koeffizient mit 2^{27} skaliert um die Eingabe der anderen Koeffizienten als Integer zu erlauben.

Z.B. in Octave sind die Koeffizienten für einen zweit-rangigen Butterworth-Filter folgendermaßen:

```
[b,a] = butter(2,0.1)
```

Die Koeffizienten sehen dann folgendermaßen aus:

```
B0 = 0.020083
B1 = 0.040167
B2 = 0.020083
A1 = -1.56102
a2 = 0.64135
```

Wenn nun die Filtergleichung der POSYS® mit der Filtergleichung von Octave verglichen wird, bemerkt man einen kleinen Unterschied dahingehend, dass die ax Komponenten in Octave subtrahiert werden, wohingegen sie in der POSYS® addiert werden. Das führt dazu, dass die a1 und a2 Koeffizienten von Octave (oder Matlab) mit -1 multipliziert werden müssen, ehe sie an die POSYS® gesendet werden können.

Das Ergebnis sieht dann folgendermaßen aus:

```
b0 = 0.020083
b1 = 0.040167
b2 = 0.020083
a1 = 1.56102
a2 = -0.64135
```

Diese Werte werden skaliert und als Ausgangsfilterkoeffizienten gesetzt, sobald der Ausgangsskalierungsfaktor K bestimmt wurde.

Bestimmung des Bi-Quad Skalierungsfaktors

Um höchste Ausgabegenauigkeit zu erlangen, sollte der programmierbare Skalierungsfaktor K so gewählt werden, dass er mit dem größten absoluten Wert der dem Wert 32767 (größte, positive Wert für ein 16-Bit Integer mit Vorzeichen) am nächsten ist, aus dem Satz von Koeffizienten (in diesem Fall a1) skaliert werden kann. Dann kann K folgendermaßen bestimmt werden:

$$a1 = K * 32767 * 2^{-27}$$

Umstellung der Gleichung:

Nach Umstellung:	$K = (a1 * 2^{27}) / 32767$
Einsetzen der Werte:	$K = (1.56102 * 2^{27}) / 32767$
Ergebnis:	$K = 6394$

Skalierung der Bi-Quad Koeffizienten

Nachdem der optimale K-Skalierungsfaktor ermittelt wurde, können die äquivalenten „Integer“ der Bi-Quad Koeffizienten

berechnet werden. Die Integer-Werte (B0, B1, B2, A1 und A2) können wie im folgenden Beispiel errechnet werden:

$$b0 = B0 * K * 2^{-27}$$

Umstellung der Gleichung:

$$B0 = (b0 * 2^{27}) / K$$

$$B0 = (1.020083 * 2^{27}) / 6394$$

$$B0 = 422$$

Bei der Verwendung dieser Formel auf alle Koeffizienten werden folgende Werte ermittelt:

B0 = 422; B1 = 843; B2 = 422; A1 = 32767; A2 = -13463

und K = 6394

Konfiguration der Bi-Quad Filter

Das Kommando SetBiQuadCoefficient wird verwendet um die Werte in die POSYS® zu laden. Das Programmierhandbuch bechreibt das Kommando etwas detaillierter. Als Beispiel um den Filter0 des Bi-Quad Filters zu programmieren, kann folgende Sequenz dienen:

SetBiQuadCoefficient Axis1, 0, 0, 0	// Deaktivierung des Filters
SetBiQuadCoefficient Axis1, 6, 0, 6396	// K definieren
SetBiQuadCoefficient Axis1, 1, 0, 422	// B0 definieren
SetBiQuadCoefficient Axis1, 2, 0, 843	// B1 definieren
SetBiQuadCoefficient Axis1, 3, 0, 422	// B2 definieren
SetBiQuadCoefficient Axis1, 4, 0, 32767	// A1 definieren
SetBiQuadCoefficient Axis1, 5, 0, -13463	// A2 definieren
SetBiQuadCoefficient Axis1, 0, 0, 1	// Filter aktivieren

Nach Komplettierung dieser Sequenz verändert sich die Ausgabe des Filters auf Achse 1 sofort.

¤

Output Limit

Die Motorausgabebegrenzung hält die Filterausgabe davon ab, eine bestimmte Spannungsvorgabe für jede Richtung an den Verstärker auszugeben bzw. zu überschreiten. Wenn der Filter einen Wert, größer als die Grenze, erzeugt, nimmt der Motorbefehl die programmierten Grenzwerte. Mit SetMotorLimit wird der Grenzwert programmiert. Mit GetMotorLimit wird der programmierte Wert gelesen. Der spezifizierte Wert ist eine 16-Bit vorzeichenlose Zahl von 0 bis 32767. Der spezifizierte Wert ist der größtmögliche Wert, der an den Verstärker/Motor ausgegeben wird. Zum Beispiel wenn die Motorgrenze auf 30000 gesetzt wurde (oder 91.6 % es maximalwertes), werden Motorausgabewerte die größer als 30000 wären, dann als 30000 ausgegeben, und Motorausgabewerte weniger als -30000 werden als -30000 ausgegeben.

Ein Output Limit kann nur im Closed Loop Mode programmiert werde, da im Open Loop Mode die Register direkt beschrieben werden.

Der Standardwert der nach einem Reset bzw. PowerOn aktiv ist, ist 100% bzw 32767.

¤

Motor Bias

Motor Bias

Wenn eine Achse abhängig ist von einer netto-bedingten von aussen wirkenden Kraft die in einer Richtung zieht/drückt (wie eine senkrechte abwärts von der Schwerkraft gezogenen Achse), kann der Servofilter eine Kompensation vornehmen, indem er eine ständige Vorgabespannung an den Filter schickt die beiträgt der Kraft entgegen zu wirken. Das Kommando hierfür heisstSetMotorBias Der programmierte Wert kann mitGetMotorBiasgelesen werden.

Solange der Regelkreis geschlossen ist, wirkt der MotorBias zu allen Zeiten. Falls der Position Loop deaktiviert ist, aber der Bahngenerator weiterhin aktiv ist, wirkt auch in diesem Fall der MotorBias.

Wenn die Regelschleife und der Bahngenerator deaktiviert werden, wird der MotorBias dann nur nach einem Übergang auf diesen Status angewandt, und eine nachfolgende Änderung des Motor-Befehls-Registers wird ohne MotorBias angewandt. Für weitere Informationen sehen Sie bitte auch im Kapitel "Aktivieren und Deaktivieren von Kontrollmodulen" nach.

Zum Beispiel, falls ein MotorBias mit einem Wert von +1000 (=~+3%) gesetzt wurde, in dem Moment in dem das KommandoSetOperatingModeausgeführt wird, um den Bahngenerator und die Regelschleife zu deaktivieren, oder zu dem Zeitpunkt bei dem eine sicherheitsrelevante Aktion ausgeführt wird, wie die Erkennung eines Schleppabstandsfehlers und diese Module sind automatisch deaktiviert, wird die Vorgabespannung von +1000 erhalten bleiben. Falls der Anwender bei dem Versuch diesem Prozess zu bereinigen einen neuen Vorgabewert von +2000 vorgibt, wird dieser Wert unabhängig vom MotorBiaswert ausgegeben.

Der Standardwert ist null (0).

Falls der spezifizierte Wert für eine Motorvorgabespannung nicht korrekt gesetzt wurde um eine externe Kraft zu kompensieren, kann es sein dass die Achse in die eine oder andere Richtung plötzlich dav onläuft, nachdem das KommandoSetOperatingModeausgeführt wurde. Es liegt in der Verantwortung des Anwenders einen MotorBiaswert zu wählen der eine sichere Operation gewährleistet.

Das KommandoSetOperatingModeist nur auf POSYS® 1800/1900 verfügbar mit F irmwareversion >2.x.

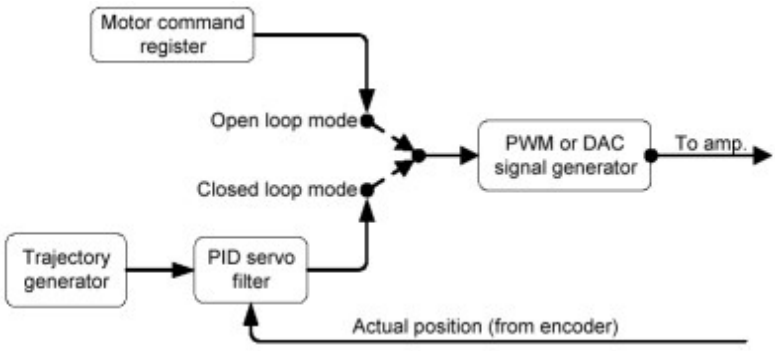
¤

Closed Loop and Open Loop Control Modes

Closed-Loop und Open-Loop Control Modi

In einem vorherigen Abschnitt wurde der Motormodus beschrieben. Für alle POSYS® hat das Ausschalten des Motors (SetMotorMode OFF) die Wirkung, das der Bahngenerator ausgeschaltet wird.

Zudem, wenn bei den Servoversionen der Motor ausgeschaltet wird, oder wenn die Abschaltung aufgrund eines Schleppabstandfehlers der höher als erlaubt war automatisch durch die POSYS® geschieht, wird die POSYS® (bzw. entsprechende Achse) in den Open-Loop-Modus geschaltet. Im Open-Loop-Modus arbeitet der Servofilter nicht und der Wert für das Motorausgangssignal muss mit dem Kommando SetMotorCommand manuell gesetzt werden. Wenn der Motor eingeschaltet ist, dann ist die Achse im Closed-Loop-Mode und der Wert für das Motorausgangssignal wird automatisch durch den Servofilter gesetzt.



Figur 3-1. Definition der Wege für das Motorausgangssignal

Der Closed-Loop Modus ist die normale Betriebsart der Servoversion der POSYS®. Der Open-Loop Modus wird typischerweise dann benutzt, wenn eine oder mehrere Achsen nur Drehmomentkontrolle benötigen, oder wenn der Verstärker kalibriert werden muss.

ANMERKUNG:
Endschalter funktionieren nicht im Open-Loop Modus.

Motor Bias in Open-Loop Modus

Die Motorvorspannung funktioniert zu allen Zeiten im Closed-Loop Modus. Wenn die Achse in den Open-Loop Modus geschaltet wird, wird weiterhin der Bias-Wert auf den Motor ausgegeben um zu verhindern, dass die Achse plötzlich in Richtung der anstehenden externen Kraft zu drücken beginnt. Sobald der Host jedoch einen neuen Motorbefehl ausgibt, ersetzt sein Wert allerdings den Vorspannungsausgang, der keine Wirkung mehr hat. Sobald die Achse zurück in den Closed-Loop Modus zurückspringt, wird der bisherige Vorspannungswert wieder aktiv.

ANMERKUNG:
Wenn der angegebene Vorspannungswert nicht richtig auf die externe Kraft ausgleichend wirkt, kann sich die Achse plötzlich in die eine oder andere Richtung nach einer SetMotorMode OFF-Anweisung bewegen. Es liegt in der Verantwortung des Anwenders, einen Motorvorspannungswert zu selektieren, der einen sicheren Betrieb gewährleistet.

¤

Piezo (Keramik) Motorfunktionen

Spezielle Kommandos für Piezo (Keramik) Motoren

Die POSYS ®-Serie 1800-Piezo und Piezo-1900 sind Positioniersteuerkarten (PC 104, PCI-und Standalone) für Servo-und Schrittmotoren und sind für 1 bis 4 Achsen ausgelegt. Dies ist eine angepasste Version mit einer speziellen Firmware. Die Funktionalität dieser Serie wurde erheblich verbessert durch die Verwendung eines speziellen Motion-C ontrol-Chips für Piezo (Keramik) Motoren um z.B. die Haftreibung zu kompensieren.

Die Piezo-Motor-Funktionalität ist auf eine bestimmte Achse aktiv, wenn der eingestellte Motortyp bürstenbehaftete oder bürstenlose Servomotor ist. Alle Standard POSYS®-Kommandos stehen weiterhin zur Verfügung, mit Ausnahme der unten aufgeführten. Dieses Dokument beschreibt die Erweiterungen und Änderungen zu dieser Version. Für genaue Angaben über die Kommandos folgen Sie bitte den Link zu Kapitel mit den einzelnen Kommandos

Benutzerdefinierte F unktionen:

Die Piezo-Motor-Funktionalität in dieser Firmware konzentriert sich um die Fähigkeit den Wert der Servo-Parameter "on the fly" als Funktion aus Schleppabstandsfehler, verbleibende Wegstrecke, oder ob sich die Achse in Bewegung befindet, anzupassen. Darüber hinaus kann der Anwender ein Offset zur Regelschleife hinzufügen. Die Verbesserungen und Vorgehensweise ist im Folgenden erklärt und aufgeführt.

1. Zwei Bedingungen müssen vorliegen um das Piezo-Motor-Verhalten an einer bestimmten Achse zu aktivieren. Der eingestellte Motortyp muss "Servo" sein (bürstenbehaftet oder bürstenlos) und der Wert PzWindow muss so eingestellt sein, dass sein Wert ungleich Null ist. Andernfalls, können nur die Standardfunktionen verwendet werden und die folgenden Schritte 2 bis 6 finden keine Anwendung.
2. Alle Regelparameter (kp, ki, kvff, etc) werden sofort aktualisiert. Das Update-Kommando ist nicht erforderlich um diese Werte zu ändern und hat keinen Einfluss auf sie.
3. Wenn der Motor in Position ist, wird der Wert für Integration Limit durch einen anderen Wert ersetzt. Der Wert für den neuen Integration Limit wird durch das neue Kommando Set/GetpzIntLimStl definiert. Dieser Wert wird nur verwendet, wenn die Achse stillsteht. Wenn die Achse in Bewegung ist, dann hängt der Wert für den Integration Limit von weiteren Faktoren ab (siehe #5). Der gültige Wertebereich für Set/GetpzIntLimStl ist der Gleiche wie für Set/GetIntegrationLimit
4. Wenn der Motor nicht in Position ist, wirkt eine von zwei Offset-Werten zusätzlich auf die Regelschleife (zusätzlich um einen Ausgleich zum Standard-Drehmoment zu bewirken, falls eingestellt). Welcher Offset verwendet wird, hängt von mehreren Bedingungen ab. Falls die verbleibende Wegstrecke größer ist, als mit Set/GetpzWindow definiert wurde, wird ein positiver Offset verwendet für positive Geschwindigkeitsprofile und ein negativer Offset für negaitve Geschwindigkeitsprofile Andernfalls wird der positive Offset verrwendet, wenn der Schleppabstandsfehler positiv ist und der negative Offset verwendet, wenn der Schleppabstandsfehler negativ ist.
   Beide Offset-Werte werden als positive Zahlen programmiert (von 0 bis 32k), aber der negative Offset wird immer als negativer Wert behandelt. Diese Offsets werden durch die Kommandos Set/GetpzOffsetPos und Set/GetpzOffsetNeg programmiert.
5. Wenn der Motor in Bewegung ist (nicht in Position), wird die verbleibende Wegstrecke im Bewegungsprofil überprüft. Falls der Wert größer ist als der Wert der mit Set/GetpzWindow programmiert wurde, wird ein völlig anderer Satz Regelparameter (Piezo-Regelparameter) aktiv. Andernfalls wird der Satz Regelparameter immer noch verwendet, wenn der Schleppabstandsfehler größer ist als pzWindow. Wenn keiner dieser Fälle zutreffen, werden die Standard-Regelparameter aktiv. Weitere Befehle wurden hinzugefügt, damit der Anwender festgelegen kann, welche neuen Parameter für dieses Fenster Anwendung finden. Die neuen Befehle sind Set/GetpzKp Set/GetpzKi Set/GetpzKd Set/GetpzKvff Set/GetpzKaff Set/GetpzIntLim und Set/GetpzKdPeriod Der neue Befehl, der verwendet wird für die neuen Regelparameter aktiv zu werden, wird durch die Definition der Schwelle für den Schleppabstandsfehler bestimmt und heisst Set/GetpzWindow
6. Ein zweiter Schleppabstandsfehlerfenster wurde ebenfalls hinzugefügt. Der Wert des sekundären Fensters wird definiert durch das Kommandopaar Set/GetpzWindow2 Wenn der Motor nicht in Position ist, und der absolute Schleppabstandsfehler ist größer als der Wert der durch dieses Kommando definiert wurde, dann wird ein neuer Velocity Feedforward-Wert für dieses sekundäre Fenster verwendet, der mit dem Kommando Set/GetpzKvff2 programmiert wird.

Der Pseudo-C ode unten erklärt das Verhalten. Dass der Motor sich In-Position befindet ist wahr, wenn Bit 7 des GetActivityStatus gesetzt ist. Dieses Bit wird nur gesetzt, wenn die Bedingungen Set/GetSettleWindow und Set/GetSettleTime erfüllt sind. SetSettleTime definiert die Länge der Zeit in Update-Rates in der sich die Position innerhalb eines SettleWindows befinden muss ehe Bit 7 in GetActivityStatus gesetzt wird.

Pseudo-Code:

```
If (motor type is "servo") and (pzWindow > 0)
{
```

```
    The UPDATE command is not needed to make a buffered position loop parameter (Kp, Kd, Ki, etc.) active.

    If ( settled )      //the conditions of SetSettleWindow and SetSettleTime are met
    {

        Use the standard set of loop gains except for integration limit
        Use the new 'in position' integration limit
        Don't apply any extra offset to the motor output

    }
    else
    {
          if(abs(DestintationPos-C ommandedPos)<pzWindow)
          {
                Apply the extra offset to the motor output based on sign of position error

                if( abs(position error) < pzWindow ) use standard set of loop gains.
                else use Piezo loop gains.

          }
          else
          {

                Apply the extra offset to the motor output based on the sign of
                CommandedVelocity
                Use Piezo loop gains.

          }

          if(abs(position error) > pzWindow2) use PzKvff2 instead of PzKvff.

    }

}
else
      use standard behavior
```

Verwendung der Kommandos:

Die Nutzung der neuen benutzerdefinierten Befehle steht im Einklang mit der Verwendung der Standard-Befehle. Zum Beispiel das Kommando SetpzKp hat die gleiche Verwendung in Bezug auf die Paket-Format, Umfang und Parameter Return-Codes, wie das Kommando SetKp. Der einzige Unterschied wäre der Opcode des Kommandos.

Alle Werte der neuen Kommandos sind standardmäßig auf Null gesetzt.

Folgende Liste gibt eine Übersicht der neuen Kommandos.

COMMAND	SIZE	OPCODE	DESCRIPTION
GetpzOffsetPos	uint16	0x35	Servo offset used when not settled and position error is positive
GetpzOffsetNeg	uint16	0x3D	Servo offset used when not settled and position error is negatiove
GetzIntLimStl	uint32	0x04	integration limit (32-bit) used when settled
GetpzWindow	uint16	0x05	Location of primary Window boundary
GetpzKp	uint16	0x08	Proportional term when position error is in primary window.
GetpzKi	uint16	0x09	Integral Term " "
GetpzKd	uint16	0x0A	Derivative Term " "
GetpzKvff	uint16	0x0B	Velocity Feed Forward Term " "
GetpzKaff	uint16	0x0C	Acceleration Feed Forward Term " "
GetpzIntLim	uint32	0x0D	Integration Limit " "
GetpzKdPeriod	uint16	0x0E	Derivative Time " "
GetpzWindow2	uint16	0x4E	Location of secondary Window boundary
GetpzKvff2	uint16	0x4F	Velocity Feed Forward Term when position error is in secondary window.
SetpzOffsetPos	uint16	0x16	See above for corresponding "Get" command.

SetpzOffsetNeg	uint16	0x17	"	•"
SetzIntLimStl	uint32	0x18	"	•"
SetpzWindow	uint16	0x19	"	•"
SetpzKp	uint16	0x1B	"	•"
SetpzKi	uint16	0x1C	"	•"
SetpzKd	uint16	0x1F	"	•"
SetpzKvff	uint16	0x20	"	•"
SetpzKaff	uint16	0x23	"	•"
SetpzIntLim	uint32	0x24	"	•"
SetpzKdPeriod	uint16	0x28	"	•"
SetpzWindow2	uint16	0x29	"	•"
SetpzKvff2	uint16	0x2E	"	•"

Korrespondierende Delphi-Typen: Word = uint16, LongWord = uint32

Diese Tabelle ist nur zur Information. Alle Kommandos (DLL) verwenden die Beschreibung wie sie in der Spalte COMMAND erscheint.

Die folgenden Kommandos wurden ersetzt durch die neuen Kommandos und sind daher nicht mehr verfügbar:

Set/GetBiQuadCoefficient
Set/GetAuxiliaryEncoderSource
Set/GetSPIMode
Set/GetPWMFrequency

Falls irgendeines der obigen Kommandos zusätzlich zu den Piezo-Kommandos benötigt wird, wenden Sie sich an servo Halbeck GmbH & Co.KG wegen einer modifizierten Firmware-Version.

Date: 01 September 2008

servo Halbeck GmbH & Co. KG

www.halbeck.com

¤

Status Register

Überblick

Überblick

Die POSYS® kann fast jeden Aspekt einer Achse überwachen. Es gibt zahlreiche numerische Register, die abgefragt werden können, um die Aktualität der Achse wie die Ist-Position (GetActualPosition-Kommando), die Soll-Position (GetCommandedPosition-Kommando) etc. zu bestimmen. Außer diesen numerischen Registern gibt es drei bit-orientierte Zustandsregister, die einen kontinuierlichen Bericht über den Status einer spezifischen Achse zurückmelden (GetActivityStatus, GetSignalStatus, GetEventStatus).

Der Host kann diese drei Register abfragen, oder die Inhalte dieser Register können in Breakpointoperationen benutzt werden, um ein Triggerereignis wie "trigger wenn Bit 8 im Zustandsregister auf "low" geht zu definieren. Diese Register liefern auch die Quelldaten für den AxisOut-Mechanismus, der einem Bit innerhalb dieser drei Register ermöglicht, als ein Hardwaresignal ausgegeben zu werden.

¤

Event Status Register

Event Status Register

Das Ereignis-Zustandsregister (Event Status Register) ist konstruiert, um Ereignisse zu registrieren, die sich nicht kontinuierlich ändern, aber dazu neigen bei einem spezifischen Ereignisse vorzukommen. Als solcher, wird jedes der Bits im Register von der POSYS® gesetzt und vom Host gelöscht.

Das Ereignis-Zustandsregister wird in der Tabelle unten beschrieben:

Bit	Name	Beschreibung
0	Motion complete	Wird gesetzt wenn ein Profil abgeschlossen wird. Die Bewegung, die für vollendet erachtet wird, kann auf der Soll-Position oder Ist-Position basieren
1	Position wrap	Wird gesetzt wenn die Ist-Position 7FFFFFFFh überschritten (die positivste Stellung) wird und konvertiert zu 80000000h (die negativste Stellung), oder umgekehrt.
2	Breakpoint 1	Wird gesetzt wenn Breakpoint #1 getriggert wird
3	Capture received	Wird gesetzt, wenn der high-speed Latch Input eine neue Positionsangabe erhält.
4	Motion error	Wird gesetzt wenn die Ist-Position von der Soll-Position mehr differiert als d Schleppabstandswert erlaubt.
5	Positive limit	Wird gesetzt wenn der positive Endschalter angefahren wird.
6	Negative limit	Wird gesetzt wenn der negative Endschalter angefahren wird.
7	Instruction error	Wird gesetzt wenn ein Kommandofehler auftritt
8, 9, 10	Reserved	Darf 1 oder 0 sein
11	Commutation error	Wird gesetzt wenn ein Kommutierungsfehler eintritt (nicht bei allen Versione gültig)
12,13	Reserved	Darf 1 oder 0 sein
14	Breakpoint 2	Wird gesetzt wenn Breakpoint #2 getriggert wird
15	Reserved	Darf 1 oder 0 sein

Das Kommando GetEventStatus liefert die Inhalte des Ereignis-Zustandsregisters für die angegebene Achse.

Bits im Ereignis-Zustandsregister werden gelatcht. Sobald gesetzt, bleiben sie gesetzt bis sie vom einem Hostkommando oder einem Systemreset gelöscht werden. Ereignis-Zustandsregisterbits können vom Kommando ResetEventStatus auf 0 zurückgesetzt werden, indem eine 16-Bit Maske benutzt wird. Register-Bits, die 0 in der Maske entsprechen, werden resetted; alle anderen Bits bleiben unbeeinflusst.
Das Ereigniszustandsregister kann auch benutzt werden, einen Interrupt zu generieren.

¤

Instruktionsfehler

Instruktionsfehler

Bit 7 des Ereigniszustandsregisters zeigt einen Anweisungsfehler an. Solch ein Fehler kommt vor, wenn eine sonst zulässige Anweisung oder Anweisungsfolge gesandt wird, wenn der gegenwärtige operative Zustand der POSYS® die Anweisung ungültig macht. Anweisungsfehler kommen immer nur zum Zeitpunt eines Updates vor. Sollte ein Anweisungsfehler auftreten, werden die ungültigen Parameter ignoriert, und der Anweisungs-Fehler-Indikator des Ereignis-Zustandsregisters wird gesetzt. Während ungültige Parameter während des Updates überprüft und ignoriert werden, werden zulässige Parameter weiterhin geschickt und akzeptiert. Dies kann unabsichtliche Nebenwirkungen je nach Art der Bewegung nach sich ziehen, sodass alle Anweisungsfehlerereignisse sehr ernst genommen werden sollten.

Beipsiel: in der folgenden Sequenz:
SetProfileMode (axis2, Velocity)
SetVelocity (axis2, -4387)
SetProfileMode (axis2, Trapezoidal)
Update (axis2)

Die negative Geschwindigkeit ist nicht zulässig in dem neuen Profilmodus. Das Update wird ausgeführt, aber das Anweisungs-Fehlerbit wird gesetzt. (legitime Parameter, wie Position, werden aktualisiert, und die Profilgenerierung arbeitet weiter).

¤

Activity Status Register

Activity Status Register

Wie das Ereignis-Zustandsregister geht das Aktivitäts-Zustandsregister verschiedenen POSYS®-Feldern nach.

Aktivitäts-Zustandsregisterbits werden allerdings nicht gelatcht, sie werden kontinuierlich von der POSYS® gesetzt und zurückgesetzt um den gegenwärtigen Status anzuzeigen.

Das Activity Status Register wird in der Tabelle unten beschrieben:

<table>
<tr><th>Bit</th><th>Name</th><th>Beschreibung</th></tr>
<tr><td>0</td><td>Reserved</td><td>n.a.</td></tr>
<tr><td>1</td><td>At maximum velocity</td><td>Gesetzt (1) wenn die Sollgeschwindigkeit gleich der Istgeschwindigkeit ist, wi vom Host angegeben. Gelöscht (0), wenn es nicht der Fall ist. Dieses Bit funktioniert nur im Trapez-, Geschwindigkeits-, oder S-Kurvenmodus. Es funktioniert nicht im Electronic-Gearmodus</td></tr>
<tr><td>2</td><td>Position Tracking</td><td>Gesetzt (1) wenn der Servo die Achse innerhalb des Tracking Windows behäl Gelöscht (0), wenn es nicht der Fall ist.</td></tr>
<tr><td>3, 4, 5</td><td>Current Profile Mode</td><td>Diese Bits zeigen den gegenwärtigen Profilmodus an, der anders sein könnte der Wert, der benutzt wurde mit dem Kommando SetProfileMode, nach dem Update-Befehl noch nicht erfolgte.
<table>
<tr><th>bit 5</th><th>bit 4</th><th>bit 3</th><th>Profile Mode</th></tr>
<tr><td>0
0
0
0
1</td><td>0
0
1
1
0</td><td>0
1
0
1
0</td><td>trapezoidal
velocity contouring
s-curve
electronic gear
external profile mode</td></tr>
</table></td></tr>
<tr><td>6</td><td>reserved</td><td>Darf 1 oder 0 sein</td></tr>
<tr><td>7</td><td>Axis-settled</td><td>Gesetzt (1) wenn sich die Achse innerhalb des SettleWindows für eine vorher spezifizierte Zeitspanne aufhielt. Gelöscht (0) wenn dies nicht der Fall ist.</td></tr>
<tr><td>8</td><td>Motor mode</td><td>Gesetzt (1) wenn der Motor "ON" ist, gelöscht (0) wenn er "OFF" ist. Wenn de Motor "ON" ist, bedeutet dies dass die POSYS® Trajektorieoperationen</td></tr>
</table>

Bit	Name	Beschreibung
		ausführen kann, und für die Servoversionen bedeutet es, daß der Regler sich Closed-Loop-Modus befindet und am regeln ist. Wenn der Motor "OFF" ist, bedeutet dies daß Trajektorieoperationen nicht ausgeführt werden können, u für die Servoversionen bedeutet es, daß der Regler sich im Open-Loop-Modu befindet und nicht regelt. Der SetMotorMode-Befehl wird normalerweise benu um den Modus des Motors zu selektieren, allerdings wird die POSYS® resette und den Modus auf 0 stellen und den Motor ausschalten falls ein Schleppabstandsfehler auftritt.
9	Position capture	Gesetzt (1) wenn ein neuer Positionswert verfügbar ist, der von der high-spe Latch Input Hardware gelesen werden kann. Gelöscht (0) wenn ein neuer We noch nicht erfasst wurde. Während dieses Bit gesetzt ist, werden keine neuer Werte erfasst. Der Befehl GetC aptureValue erfasst den Wert und löscht das Damit ist die nächste Erfassung möglich.
10	In-motion indicator	Gesetzt (1) wenn die Soll-Position des Trajektorieprofils sich ändert. Gelöscht (0) wenn die Soll-Position sich nicht ändert. Der Wert dieses Bits kann oder d nicht dem Wert des Motion C omplete Bits des Ereigniszustandsregisters entsprechen, unabhängig davon ob der Motion C omplete Mode auf Soll- oder Ist-Wert gesetzt wurde.
11	In positive limit	Gesetzt (1) wenn der Motor den positiven Endschalter erreicht hat. Gelöscht (wenn dem nicht so ist.
12	In negative limit	Gesetzt (1) wenn der Motor den negativen Endschalter erreicht hat. Gelöscht (0) wenn dem nicht so ist.
13, 14, 15	S-curve segment	Zeigt die Nummer der einzelnen Segmente der S-Kurve mit den Werten 1-7 wie gezeigt in der S-Kurvenbeschreibung dieses Handbuches. Ein Wert von 0 diesem Feld zeigt, dass die Bahn nicht in Bewegung ist. Dieses Feld ist undefiniert für andere Profilmodi als der S-Kurve und kann 0's oder 1's enthalten.

Das Kommando GetActivityStatus liefert die Werte des Aktivitäts-Zustandsregisters für die angegebene Achse.

¤

Signal Status Register

Signal Status Register

Das Signal Status Register bietet Real-Time Signalerfassung für verschiedene Eingänge. Das Signal Status Register wird in der Tabelle unten bschrieben:

Bit	Name	Beschreibung
0	A Encoder	A-Signal des Enkodereingangssignals
1	B Encoder	B-Signal des Enkodereingangssignals
2	Index Encoder	Index-Signal des Enkodereingangssignals
3	Home	Referenzschaltereingang
4	Positive Limit	Positiver Endschaltereingang
5	Negative Limit	Negativer Endschaltereingang
6	AxisIn	Frei-programmierbarer achsenspezifischer Eingang
7	Not for use	with Servo or Stepper version
8	Not for use	with Servo or Stepper version
9	Not for use	with Servo or Stepper version
10	AxisOut	Frei-programmierbarer achsenspezifischer Ausgang
11-15	Reserved	

Das Kommando GetSignalStatus liefert den Inhalt des Signal Status Registers der spezifizierten Achse. Alle Signal Status Register Bits sind Eingänge mit Ausnahme von Bit 10 (AxisOut).
Die Bits in den Signal Status Register stellen immer das eigentliche Hardwareniveau auf den entsprechenden Eingängen dar. Eine ''1'' in diesem Register stellt einen elektrisch hohen Wert auf dem Eingang dar; eine ''0'' zeigt ein elektrisch niedriges Niveau an. Der Zustand der Signal Sense Mask beeinflusst den Wert der gelesen wurde mit dem Kommando GetSignalStatus (lesen Sie bitte im nächsten Abschnitt mehr über die Signal Sense Mask).

¤

Signal Sense Mask

Signal Sense Mask

Die Bits in dem Signal Status Register stellen den high/low Zustand von verschiedenen E/As auf der POSYS® dar. Wie diese E/As von der POSYS® interpretiert werden, kann durch die Verwendung der Signal Sense Mask definiert werden. Dies ist nützlich um die Interpretation von Eingangsignalen zu verändern um sich wiederum der Interpretation der Signale der Hardware des Anwenders anzupassen.

Das SignalSenseMask-Register wird in der Tabelle unten beschrieben:

Bit	Name	Interpretation
0	A encoder	Gesetzt (1) um A-Enkodereingangssignal zu invertieren. Nicht gesetzt (0) keine Invertierung.
1	B encoder	Gesetzt (1) um B-Enkodereingangssignal zu invertieren. Nicht gesetzt (0) keine Invertierung.
2	Index encoder	Gesetzt (1) um Indexenkodersignal zu invertieren. Nicht gesetzt (0) für keine Invertierung.
3	Home	Gesetzt (1) um Referenzsignal zu invertieren. Nicht gesetzt (0) für keine Invertierung.
4	Positive Limit	Gesetzt (1) für eine "active high" Interpretation des positiven Endschalters welches bedeutet, dass der Endschalter für die positive Fahrtrichtung angefahren wurde, wenn das Signal auf "active high" geht. Kein Endschalt (0) wenn "active low".
5	Negative Limit	Gesetzt (1) für eine "active high" Interpretation des negativen Endschalter welches bedeutet, dass der Endschalter für die negative Fahrtrichtung angefahren wurde, wenn das Signal auf "active high" geht. Kein Endschalt (0) angefahren wenn "active low".
6	AxisIn	Gesetzt (1) um das AxisIn Signal zu invertieren. Nicht gesetzt (0) für kein Invertierung.
7-9	n/a	keine Verwendung
10	AxisOut	Gesetzt (1) um das AxisOut Signal zu invertieren. Nicht gesetzt (0) für kei Invertierung.
11-15	reserviert	

Das Kommando SetSignalSense setzt den Wert für die Signal Sense Maske. Das Kommando GetSignalSense liefert die gegenwärtige Einstellung.

¤

Überwachung des Bewegungsverhaltens

Host Interrupts

Mit Interrupts ist es möglich den Host auf spezielle Ereignisse aufmerksam zu machen ohne die Notwendigkeit kontinuierlich oder durch Polling die Ereignisregister zu überwachen. Die POSYS® stellt diese Möglichkeit als eine Form des Hostinterrupt zur Verfügung.
Die Ereignisse die einen Hostinterrupt auslösen sind die Selben die den Bits im "Event Status Register" zugewiesen wurden - hier noch einmal zur Erläuterung aufgelistet:

Bit	Ereignis	Passiert wann
0	Motion complete	Das Profil erreicht seinen Endpunkt, oder die Bewegung wurde anderwei zu einem Stop veranlasst.
1	Wrap-around	Positionszählerüberlauf mit Addierung
2	Breakpoint 1	Breakpoint 1 Bedingung eingetreten
3	Capture received	Enkoder- bzw. Homeschalterindex erfasst
4	Motion error	Der maximale Schleppfehlerabstand wurde für eine bestimmte Achse überschritten
5	In positive limit switch	Positiver Endschalter angefahren
6	In negative limit switch	Negativer Endschalter angefahren
7	Instruction error	Hostkommando verursacht Fehler
11	Commutation Error	Der Indextakt steht nicht im Einklang mit der aktuellen Phase (nur für POSYS® 1800/1900 mit Firmwareversion >2.x)
14	Breakpoint 2	Breakpoint 2 Bedingung eingetreten

Bei der Benutzung einer 16-Bit Maske, kann der Host ein oder alle dieser Bits bedingen, um eine Unterbrechung zu verursachen. Diese Maske wird gesetzt mit dem Kommando SetInterruptMask. Der Wert der Maske kann zurückgelesen werden mit dem Kommando GetInterruptMask. Die Bitpositionen der Maske entsprechen den Bitpositionen des "Event Status Register". Wenn eine 1 in der Maske gespeichert wird, wird eine 1 in dem entsprechenden Bit des "Event Status Register" einen Interrupt auslösen. Jede Achse unterstützt seine eigene Interruptmaske. Dies erlaubt für jede Achse eine unterschiedliche Interruptbedingung zu setzen.

Die POSYS® fragt immer wieder das "Event Register" und die Interruptmaske ab um festzustellen ob ein Interrupt ausgelöst wurde oder nicht. Wenn ein Interrupt ausgelöst wurde, wird das Hostinterrupt-Signal aktiv.

An diesem Punkt kann der Host auf das Interrupt reagieren (obwohl die Ausführung der momentanen Hostanweisung, inklusive des Transfers aller damit verbundenen Datenpakete abgeschlossen sein sollte), aber es ist nicht unbedingt erforderlich.

Da es für mehr als eine Achse möglich ist, konfiguriert zu werden, Interrupts gleichzeitig zu generieren, stellt die POSYS® das Kommando GetInterruptAxis zur Verfügung. Dieses Kommando liefert einen bitmaskierten Wert mit einem gesetzten Bit für jede gegenwärtig eine Unterbrechung generierende Achse zurück. Bit 0 wird gesetzt wenn Achse 1 unterbricht, Bit 1 wird für Achse 2 gesetzt, etc. Wenn kein Interrupt ausgelöst wurde, dann sind keine Bits gesetzt.

Um einen Interrupt zu bearbeiten, können normale POSYS®-Kommandos verwendet werden. Die Kommandos die vom Host gesendet werden um den Interrupt zu behandeln, hängen von der unterbrechenden Bedingung ab, jedoch muss als Mindestes das unterbrechende Bit im "Event Status Register" mit dem Kommando ResetEventStatus zurückgesetzt werden. Falls dies nicht geschieht, würde augenblicklich der selbe Interrupt ausgelöst, nachdem die Interrupts wieder aktiviert sind.

Sobald der Host die Bearbeitung des Interrupts abgeschlossen hat, sollte ein ClearInterrupt-Kommando gesendet werden, um den Interruptanschluss zu löschen und wieder frei zu machen. Anmerkung: falls ein weiterer Interrupt zur Bearbeitung ansteht wird die Interruptleitung nur für den momentanen Interrupt wieder frei gemacht und erneut belegt.

Im Folgenden ist eine typische Sequenz von Interrupts und Hostreaktionen. In diesem Beispiel hat eine Achse einen Endschalter in der positiven Richtung angefahren, die ein Endschalterereignis und einen abrupten Stopp verursachte. Der abrupte Stopp verursacht einen Schleppabstandsfehler. Nehmen wir an, dass diese Ereignisse alle mehr oder weniger gleichzeitig eintreten. In diesem Beispiel ist die Unterbrechungsmaske (interrupt mask) für diese Achse gesetzt worden, so dass entweder Schleppabstandsfehler oder Endschalterereignisse einen Interrupt verursachen.

Ereignis	Hostaktion
Schleppabstandsfehler und Endschalterereignis generieren einen Interrupt	sendet GetInterruptAxis Kommando
Eine Bitmaske, die alle unterbrechenden Achsen identifiziert, wird von der POSYS® zurückgeliefert. Dieser Wert identifiziert eine Achse als die Interupt verursachende Achse.	sendet GetEventStatus Kommando und ermittelt daß die Flags für Schleppabstandsfehler und Endschalterereignis gesetzt sind. Sendet daraufhin ein ResetEventStatus Kommando um beide Bits zurückzusetzen. Setzt mit dem Kommando SetMotorMode ON wieder die Achse in den Closed Loop Mode zurück. Ein ClearInterrupt-Kommando setzt die Interruptleitung zurück.
POSYS® setzt Schleppabstandsfehlerbit zurück und deaktiviert Hostinterruptleitung	Generiert eine negative Bewegung um vom Endschalter herunter zu fahren
Motor bewegt sich vom Endschsalter. Activity Status Endschalter Bit ist gelöscht.	Keine

Am Ende dieser Sequenz werden alle Zustandsbits gelöscht, die Interruptleitung ist inaktiv, und es stehen keine weiteren Interrupts unbearbeitet an.

¤

In-Motion Indikator

In-motion Indikator

Die POSYS® kann feststellen ob die Achse in Bewegung ist oder nicht. Dies wird durch den "In-Motion Indicator" bewerkstelligt.

Der "In-Motion Indicator" erscheint im Bit 10 im "Activity Status Register". Der "In-Motion Bit" ist ähnlich dem "Motion Complete Bit", jedoch gibt es 2 signifikante Unterschiede. Der 1. Unterschied ist (wie bei allen Bits im "Activity Status Register") der "In-Motion Indicator" zeigt kontinuierlich den Status ohne Interaktion mit dem Host an. Mit anderen Worten, der "In-Motion Bit" kann nicht durch den Host gesetzt oder zurückgesetzt werden. Der andere Unterschied ist, dass dieser Bit den Soll-Bewegungszustand des Trajektoriegenerators anzeigt und nicht den aktuellen Enkoderzustand.

Der "Motion Complete Bit" funktioniert nur im S-Kurven-, Trapez- und Geschwindigkeitsmodus. Es funktioniert nicht im Electronic Gearing Modus.

¤

Schleppabstandsfehler

Schleppabstandsfehler

Unter gewissen Umständen kann sich die Istposition (Enkoderstellung) von der Sollposition (aktuelle Sollpositionsvorgabe des Trajektoriegenerators) in erheblichem Umfang unterscheiden. Solch ein übermäßiger Positionsfehler zeigt oft einen potenziell gefährlichen Zustand wie Motor- oder Enkoderausfall, oder übermäßige mechanische Reibung an.

Um diesen Zustand zu entdecken, der die persönliche und Anlagensicherheit zu erhöhen helfen soll, verfügt die POSYS® über einen programmierbaren maximalen Schleppabstandsfehler.

Das Kommando SetPositionErrorLimit wird verwendet um den maximalen Schleppabstandsfehler zu programmieren und der programmierte Wert wird mit dem Kommando GetPositionErrorLimit ausgelesen. Um zu bestimmen, ob ein Schleppabstandsfehler vorliegt, wird kontinuierlich die Istposition mit der Sollposition verglichen. Wenn der Schleppabstandsfehlergrenzwert überschritten wird, spricht man von einem "Schleppabstandsfehler".

In dem Augenblick in dem ein Schleppabstandsfehler passiert, werden weitere Aktionen gleichzeitig gestartet. Die folgende Liste beschreibt welche Ereignisse stattfinden:

- Der "Motion Bit Error" des "Event Status Word" wird gesetzt.
- Falls "automatischer Stop" bei Schleppabstandsfehlererkennung gesetzt ist, wird der Motor ausgeschaltet, welches zur

Folge hat, dass der Trajektoriegenerator deaktiviert (alle POSYS® Versionen) und bei den Servoversionen zusätzlich vom "Closed-Loop-Mode" in den "Open-Loop-Mode" geschaltet wird. Dies ist gleich zu setzen mit dem Kommando (SetMotorMode axis, OFF).

Um einen Schleppabstandsfehler zu lösen, der zum Ergebnis hat, dass der Motor ausgeschaltet wird, sollte die folgende Sequenz ausgeführt werden:

- Ursache für Schleppabstandsfehler bestimmen und Problem korrigieren (dies kann interaktives Eingreifen voraussetzen).
- Das Kommando SetMotorMode Achse, ON) gefolgt von einem Update-Befehl verwenden.

Nach der obigen Sequenz wird die Achse in Ruheposition sein, der Motor aber eingeschaltet.

Falls "automatischer Stop bei Schleppabstandsfehler" nicht gesetzt ist, wird nur der "Motion Error Status Bit" gesetzt. Der Motor wird nicht angehalten und es ist keine weitere Aktion notwendig die Sequenz weiterlaufen zu lassen. Nichtsdestotrotz sollte aus Sicherheitsgründen die Bewegungssequenz manuell angehalten, der Motor ausgeschaltet und der Grund für den Schleppabstandsfehler ergründet werden.

Automatischer Stop beim Schleppabstandsfehler

Weil ein Schleppabstandsfehler ein ernstes Problem anzeigen kann, ist es nützlich, die Achse automatisch zu einem Stillstand kommen zu lassen, bis das Problem ergründet und korrigiert worden ist. Dieses Merkmal wird als automatischer Stopp beim Schleppabstandsfehler bezeichnet.

Das Kommando SetAutoStopMode definiert auf welche Art der Controller bei einem Schleppabstandsfehler reagieren soll. Die Optionen für diesen Befehl sind aktiv und deaktiv.

Wenn Autostop aktiv ist, dann, wenn eine Schleppabstandsfehlerüberschreitung eintritt wird ein SetMotorMode OFF Kommando generiert der die Wirkung hat sofort den Trajektoriegenerator anzuhalten und (für Servoversionen) die POSYS® in "Open-Loop-Mode" zu schalten.

Bei Achsen der POSYS® in Schrittmotormodus wird der Motor sofort aufhören sich zu bewegen, was einem AbruptStop gleichkommt. Für Achsen der POSYS® in Servomodus gilt, dass der Trajektoriegenerator sofort aufhört zu arbeiten, aber weil ein MotorOff im Servomodus bedeutet, dass der Controller in den "Open-Loop-Mode" schaltet, wird der Motor langsam entsprechend seiner vorhergehenden Geschwindigkeit und der angehängten Last zu einem Stop kommen. Bei der Servoversion kann der Übergang in den "Open-Loop-Mode" gefährlich sein, falls die Achse vertikal arbeitet, was zur Folge haben kann, dass die Achse aufgrund der Schwerkraft nach unten zieht, falls kein Feedback für die Servoregelung erfolgt. Diesem Problem kann begegnet werden wenn der Motorvorspannungswert (Set/GetMotorBias) berichtigt wurde, der in den Servo-Filter-Abschnitten dieses Dokumentes beschrieben wird, oder eine Motorbremse eingebaut wurde. Die Motorvorspannung ist ein fester Open-Loop Kommando für den Motor, der normalerweise dem PID-Filter-Ausgang hinzugefügt wird. Bei einem Schleppabstandsfehler, mit "Automatic Stop Enabled", wird die Motorvorspannung ausgegeben, sogar während der Regler im "Open-Loop-Modus" arbeitet.

Dies bedeutet, dass mit einer korrekt eingestellten Motorvorspannung, wenn die Achse einen Schleppabstandsfehler hat, der den zulässig programmierten Wert übersteigt, die Motorvorspannung erhalten bleibt, auch wenn die Achse im "Open-Loop-Mode" ist.

Vorsicht: Weil der Motorvorspannungswert für die Achse im "Open-Loop-Mode" gilt, sollte man Vorsicht walten lassen beim setzen dieses Wertes. Sie kann NICHT als Ersatz für eine Motorbremse angesehen werden.

¤

Motion Complete Indikator

Motion Complete Indikator

In vielen Fällen ist es nützlich, den Regler signalisieren zu lassen, dass eine vorgegebene Bewegung abgeschlossen wurde. Diese Funktion ist verfügbar als "Motion Complete Indicator".

Der "Motion Complete Indicator" erscheint in Bit 0 des "Event Status Register". Wie alle Bits im "Event Status Register" wird der "Motion Complete Bit" gesetzt durch den Controller und vom Host wieder zurückgesetzt. Wenn eine Bewegung abgeschlossen worden ist, setzt der Controller das "Motion Complete Bit". Der Host kann dieses Bit durch das Abfragen des "Event Status Register" prüfen oder der Host kann einige automatische Folgeaufgaben programmieren, die einen Breakpoint, einen Interrupt, oder ein AxisOut-Signal benutzen. In jedem Fall sollte der Host, sobald der Host erkannt hat, dass die Bewegung abgeschlossen worden ist, das "Motion Complete Bit" löschen, welches das Ende der Bewegung signalisiert.

"Motion Complete" kann das Ende der Bewegung in einer von zwei Arten anzeigen. Die 1. Möglichkeit ist Kommando-basiert; der "Motion Complete Indicator" wird alleine auf Basis des Trajektoriegenerators gesetzt.
Die andere Methode basiert auf die aktuelle Situation, welches bedeutet dass der "Motion Complete Indicator" auf die aktuelle Enkoderposition bezogen wird.

Das Kommando SetMotionCompleteMode bestimmt, welcher Zustand den Indikator kontrolliert. Wenn es auf den Sollwert basiert, wird die Bewegung für beendet angesehen, wenn die Sollgeschwindigkeit und -beschleunigungswerte beide null werden. Dies geschieht normalerweise am Ende einer Bewegung wenn die Sollposition erreicht worden ist. Aber es kann auch geschehen als das Ergebnis eines Stoppbefehles (SetStopMode-Befehl), einer Geschwindigkeitsänderung zu null, oder wenn ein Endschalter angefahren wird.

Wenn es auf die aktuelle Gegebenheit basiert, wird die Bewegung als beendet betrachtet, wenn alle folgenden Bedingungen erfüllt sind:

- Die Sollposition wurde erreicht
- Die Differenz zwischen der Ist-Position und der Soll-Position ist weniger als oder gleich dem mit SetSettleWindow definierten Wert. Dieser Wert kann zurückgelesen werden mit dem Kommando GetSettleWindow
- Die beiden vorhergehenden Bedingungen wurden für eine kontinuierliche Periode von X-Zyklen erfüllt, wobei X der programmierte Wert für "SettleTime" ist. Dieser Wert wird gesetzt/programmiert mit dem Kommando SetSettleTime und kann mit dem Kommando GetSettleTime zurückgelesen werden.

Am Ende des Bewegungsprofils wird der Periodenzeitgeber für den aktuell-basierten Bewegungsendemechanismus gelöscht. Dadurch wird es mindestens X Periodenverzögerungen (durch SetSettleTime definiert) geben zwischen dem Profilgenerator, der zu einem Ende gekommen ist und dem Zeitpunkt zu dem der "Motion Complete Bit" gesetzt wird.

Angemessene Programmierung muss gewährleistet sein beim aktuell-basierten Motion Complete Mode, weil es durchaus möglich ist, dass der Motion Complete Bit nie gesetzt wird, falls der Servoregelkreis nicht genau genug eingestellt wurde, um innerhalb des programmierten Positionsfehlerfensters für die spezifizierte Zeit zu verbleiben.

Der "Motion Complete Bit" funktioniert nur im S-Kurven-, Trapez- und Geschwindigkeitsmodus. Es funktioniert nicht im Electronic Gearing Modus.

¤

Settled Indikator

Settled Indikator

Die POSYS® kann auch kontinuierlich anzeigen ob oder ob nicht die Achse still steht.

Der "Settled Indicator" erscheint im Bit 6 des "Activity Status Register". Der "Settled Indicator" ist ähnlich dem "Motion Complete Bit" wenn dieser im "Actual-Modus" läuft. Die Differenzen sind dass der "Settled Indicator" kontinuerlich seinen Zustand anzeigt (kann nicht gesetzt oder gelöscht werden) und auch anzeigt ungeachtet ob oder ob nicht der "Motion Complete Mode" auf "aktuell" gesetzt wurde.

Die Achse wird für als "In-Position" gehalten, wenn die Achse steht (z.B. eine Bewegungssequenz für die Achse wird nicht ausgeführt) und wenn die aktuelle Position des Motors sich in der Soll-Position für die programmierte "SettleTime" befindet.

"SettleWindow" und "SettleTime" zusammen mit dem "Settled Indicator" sind das selbe wie "Motion Complete Bit". Entsprechend werden die gleichen Kommandos verwendet um die Werte zu setzen oder zurück zu lesen: SetSettleWindow, GetSettleWindow, SetSettleTime, GetSettleTime.

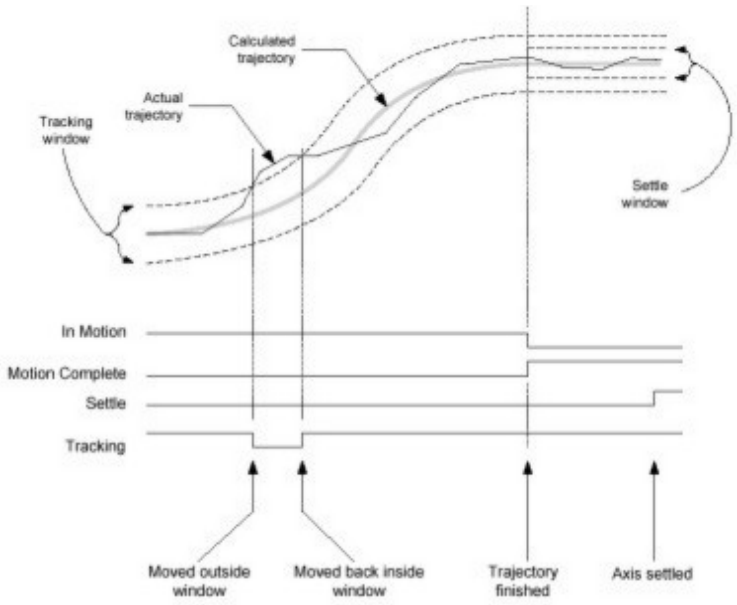


Figure 7.5-1. Das Tracking Window

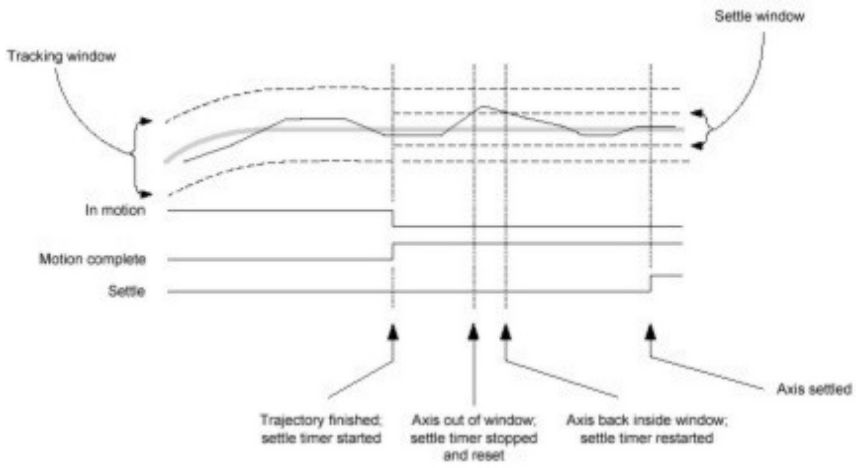


Figure 7.5-2. Das Settle Window

¤

Trace Capture

Trace Capture

"Data Trace" ist eine mächtige Funktion diverse Parameter und Register der POSYS® kontinuierlich zu überwachen und im On-Board Speicher der POSYS® zu speichern. Die ermittelten Daten können anschliessend vom Host heruntergeladen werden in dem man die Kommandos für den Zugriff auf den On-Board Speicher verwendet. "Data Trace" ist eine Funktion um die Regelparameter zu optimieren indem Vergleichswerte und Sensordaten ermittelt und mit Sollwerten verglichen werden.

Bei der Datenerfassung oder Abarbeitung von Bewegungsprofilen aus dem On-Board Speicher, wird der POSYS® zusätzliche erhebliche Arbeit abverlangt. In Applikationen in denen hoch-performante Szenarien vorkommen (z.B. Einzel-Achsenanwendung mit sich pro Updaterate verändernden Parametern), kann es unter Umständen sinnvoll sein die Updaterate zu erhöhen.

Es gibt generell 2 verschiedene Möglichkeiten der "Data Trace"-Erfassung. Die eine Möglichkeit wird durch die POSYS® bewerkstelligt, die andere durch den Host. Der Host spezifiziert welche Parameter erfasst werden sollen, und auf welche Art die Erfassung vor sich gehen soll. Die POSYS® vollzieht den "Trace" und der Host kann die Daten zur Auswertung

abrufen während (mit Dual Port RAM) oder nachdem (sinnvoller mit Single Port RAM) die Erfassung beendet wurde. Es ist aber auch möglich eine kontinuierliche Erfassung vom Host laufen zu lassen, während die POSYS® selber weiter Daten erfasst.

Um einen "Trace" zu starten, müssen eine bestimmte Anzahl Parameter definiert werden. Diese sind nachfolgend gelistet:

"Trace Buffer" - Der Host muss den Bufferspeicher für die Erfassung initialisieren. Die POSYS® stellt verschiedene Instruktionen zur Verfügung um den On-Board Speicher als Buffer zu initialisieren und erlaubt dadurch Start- und Endpunkte des On-Board Speichers für die Speicherung zu definieren.

"Trace Variables" - Es gibt je nach Modell 27 bzw. 28 verschiedene Variablen die gespeichert werden können, als da wären Istposition, Event Status Register, Schleppabstandsfehler usw. Der Anwender muss auswählen welche Variablen und von welcher Achse, erfasst werden sollen.

"Trace Period" - Die POSYS® kann in frei programmierbaren Zyklen (einzelner Zyklus, in unregelmäßigen Abständen oder mit einer bestimmten Frequenz) Daten erfassen. Diese Variable muss angegeben werden.

"Trace Mode" - 2 verschiedene Trace-Modi stehen zur Verfügung, einmal oder kontinuierlich. Diese spezifiziert auf welche Art die Daten gespeichert werden und ob der Trace-Modus automatisch stoppt oder vom Host unterbrochen werden muss.

"Trace Start/Stop Conditions" - Damit die Datenerfassung präzise und synchronisiert abläuft ist es möglich Start- und Stopp-Bedingungen zu definieren. Die POSYS® erlaubt die Überwachung der spezifizierten Konditionen und startet und stoppt die Erfassung automatisch ohne Hostintervention.

Der Trace Buffer

Die POSYS® kann On-Board Speicher in Datenbuffer organisieren. Jeder Buffer erhält eine numerische ID (Adresse). Der "Trace Buffer" muss immer ID (0) sein. Ehe die Parametererfassung verwendet werden kann, muss der Speicherbuffer 0 mit einer gültigen Adresse und Länge programmiert werden.

Die Größe des Datenbuffers definiert die maximale Anzahl möglicher zu erfassende Datenpunkte. Die maximale Größe des Datenbuffers ist wiederum nur durch die maximal zur Verfügung stehende physikalische Speicherkapazität des Systems begrenzt. Der adressierbare Speicherbereich erlaubt je nach Modell entweder 32 KB Dual-Port RAM (POSYS® 182x/192x Standard oder als Option) oder 512KB Single-Port RAM (Standard oder als Option) zu adressieren die komplett für die Speicherung von Daten verwendet werden können.

Während die Datenerfassung läuft, ist es nicht möglich die Konfiguration für die Datenerfassung zu ändern. Falls ein Versuch unternommen wird die Basisadresse, -länge oder den "Write Pointer", die mit Buffer 0 assoziiert sind, während der Erfassung zu ändern, wird dieser Versuch ignoriert und ein Error Flag gesetzt. Jedoch ist es möglich den "Read Pointer" und "Read Data" des Trace Buffers zu ändern während die Erfassung läuft. Dies erlaubt den Buffer kontinuierlich leeren zu können, während die Erfassung weiterläuft und wieder frei zu machen für die Erfassung neuer Daten.

Der Trace Period

Das Erfassungssystem unterstützt ein konfigurierbares Periodenregister. Dieser Periodenregister definiert die Frequenz innerhalb derer Daten im Trace Buffer gespeichert werden. Die Frequenz wird in Zykluseinheiten definiert, wobei ein Zyklus die minimale Periode darstellt um eine Erfassung für alle Achsen durchzuführen.

Das Kommando SetTracePeriod definiert die Länge der Periode, das Kommando GetTracePeriod liefert den programmierten Wert zurück.

Trace Variablen

Wenn eine Erfassung läuft ist es möglich zwischen 1 und 4 POSYS®-Parameter im Speicher pro Zyklus abzulegen. Die 4 "Trace Variable Registers" werden verwendet um zu definieren, welche Parameter gespeichert werden. Verwenden Sie die folgenden Kommandos für die Konfiguration:
Das Kommando SetTraceVariable definiert welche Parameter gespeichert werden sollen. Das Kommando GetTraceVariable liefert die programmierten Werte zurück.

Der Wert des Traceparameters der gesetzt und zurückgelesen wird durch die beiden vorangegangen Kommandos, definiert die Achse und die Art der Daten die gespeichert werden sollen. Das Format des Datenwortes ist wie folgt:

Bits	Name	Beschreibung
0-1	Axis	Selektiert die Quellachse für die Parametererfassung
2-7	Reserved	Muss 0 sein
8-15	ID	Selektiert die zu speichernden Parameter

Die unterstützten Parameter-ID-Werte für POSYS® 700/800/800-B/900 und POSYS® 1800/1900 (<= Firmwareversion 1.5) sind:

ID	Name	Beschreibung
0	None	Zeigt an dass keine Daten für die Tracevariable selektiert wurden
1	Position Error	Differenz zwischen Ist- und Sollposition für die spezifizierte Achse.
2	Commanded Position	Die augenblickliche Sollpositionsvorgabe durch den Regelkreis
3	Commanded Velocity	Die augenblickliche Sollgeschwindigkeitsvorgabe durch den Regelkreis
4	Commanded Acceleration	Die augenblickliche Sollbeschleunigungsvorgabe durch den Regelkreis
5	Actual Position	Die Istposition des Motors
6	Actual Velocity	Eine geschätzte Istgeschwindigkeit (kalkuliert durch einen simplen Low-Pass Filter).
7	Motor Torque	Drehmomentsollwert (Ausgabe durch den Servofilter/Regelkreis).
8	Chipset Time	Die Zykluszeit (Einheiten in Servoloop Updateraten).
9	Capture Value	Die augenblicklichen Inhalte des High Speed Capture Registers.
10	Servo Integral	Der momentane Integralwert
11	Servo Derivative	Der derivative Wert der vom Regelkreis verwendet wird.
12	Event Status	Die augenblicklichen Inhalte des Event Status Registers.
13	Activity Status	Die augenblicklichen Werte des Activity Status Registers.
14	Signal Status	Die augenblicklichen Werte des Signal Status Registers.
15	Phase Angel	Der gegenärtige Phasenwinkel (nur bürstenlose Servomotoren).
16	Phase Offset	Der gegenwärtige Phasenversatzwert (nur bürstenlose Servomotoren).
17	Phase A Output	Der Wert der gegenwärtig ausgegeben wird an die Motorwicklung 1.
18	Phase B Output	Der Wert der gegenwärtig ausgegben wird an die Motorwicklung 2. Nur gültig für 2- bzw. 3-Phasenmotoren.
19	Phase C Output	Der Wert der gegenwärtig ausgegben wird an die Motorwicklung 3. Nur gültig für 3-Phasenmotoren.
20	Analog Input 0	Der zuletzt gelesene Wert am Analogeingang 0.
21	Analog Input 1	Der zuletzt gelesene Wert am Analogeingang 1.
22	Analog Input 2	Der zuletzt gelesene Wert am Analogeingang 2.
23	Analog Input 3	Der zuletzt gelesene Wert am Analogeingang 3.
24	Analog Input 4	Der zuletzt gelesene Wert am Analogeingang 4.
25	Analog Input 5	Der zuletzt gelesene Wert am Analogeingang 5.
26	Analog Input 6	Der zuletzt gelesene Wert am Analogeingang 6.
27	Analog Input 7	Der zuletzt gelesene Wert am Analogeingang 7.
28	PID Position Error	Dies ist die Differenz zwischen den Ist- und Sollpositionswerten für die spezifizierte Achse. Dieser Wert wird für die Berechnung des Fehlerwertes für die PID-Filterberechnung verwendet.

Die unterstützten Parameter-ID-Werte für POSYS® 1800/1900 (> Firmwareversion 2.x) sind:

ID	Name	Beschreibung
0	None	Zeigt an dass keine Daten für die Tracevariable selektiert wurden
1	Position Error	Differenz zwischen Ist- und Sollposition für die spezifizierte Achse.
2	Commanded Position	Die augenblickliche Sollpositionsvorgabe durch den Regelkreis
3	Commanded Velocity	Die augenblickliche Sollgeschwindigkeitsvorgabe durch den Regelkreis
4	Commanded	Die augenblickliche Sollbeschleunigungsvorgabe durch den Regelkreis

ID	Name	Beschreibung
	Acceleration	
5	Actual Position	Die Istposition des Motors
6	Actual Velocity	Eine geschätzte Istgeschwindigkeit (kalkuliert durch einen simplen Low-Pass Filter).
7	Motor Torque	Drehmomentsollwert (Ausgabe durch den Servofilter/Regelkreis).
8	Chipset Time	Die Zykluszeit (Einheiten in Servoloop Updateraten).
9	Capture Value	Die augenblicklichen Inhalte des High Speed Capture Registers.
10	Servo Integral	Der momentane Integralwert
11	Servo Derivative	Der derivative Wert der vom Regelkreis verwendet wird.
12	Event Status	Die augenblicklichen Inhalte des Event Status Registers.
13	Activity Status	Die augenblicklichen Werte des Activity Status Registers.
14	Signal Status	Die augenblicklichen Werte des Signal Status Registers.
15	Phase Angel	Der gegenärtige Phasenwinkel (nur bürstenlose Servomotoren).
16	Phase Offset	Der gegenwärtige Phasenversatzwert (nur bürstenlose Servomotoren).
17	Phase A Output	Der Wert der gegenwärtig ausgegeben wird an die Motorwicklung 1.
18	Phase B Output	Der Wert der gegenwärtig ausgegben wird an die Motorwicklung 2. Nur gültig für 2- bzw. 3-Phasenmotoren.
19	Phase C Output	Der Wert der gegenwärtig ausgegben wird an die Motorwicklung 3. Nur gültig für 3-Phasenmotoren.
20	Analog Input 0	Der zuletzt gelesene Wert am Analogeingang 0.
21	Analog Input 1	Der zuletzt gelesene Wert am Analogeingang 1.
22	Analog Input 2	Der zuletzt gelesene Wert am Analogeingang 2.
23	Analog Input 3	Der zuletzt gelesene Wert am Analogeingang 3.
24	Analog Input 4	Der zuletzt gelesene Wert am Analogeingang 4.
25	Analog Input 5	Der zuletzt gelesene Wert am Analogeingang 5.
26	Analog Input 6	Der zuletzt gelesene Wert am Analogeingang 6.
27	Analog Input 7	Der zuletzt gelesene Wert am Analogeingang 7.
28	PID Position Error	Dies ist die Differenz zwischen den Ist- und Sollpositionswerten für die spezifizierte Achse. Dieser Wert wird für die Berechnung des Fehlerwertes für die PID-Filterberechnung verwendet.
29	Phase Angle Scaled	Der Phasenwinkel von 0°-360° anstatt in Encoderzählwerten
56	Drive Status	Die augenblicklichen Inhalte des Drive Status Registers.
57	Position Loop Integral Contribution	Dieses Register enthält den Anteil des integralen Teils der PID-Schleife
64	Biquad1 Input	Der Eingabewert für den Biquad 1 Filter
65	Biquad2 Input	Der Eingabewert für den Biquad 2 Filter

Wird der Parameter einer Variablen auf Null eingestellt, werden auch automatisch alle nachfolgenden Parameter deaktiviert. Deswegen, wenn x-Parameter während jeder Traceperiode gespeichert werden sollen, muss die Tracevariable 0 -(X-1) verwendet werden um zu identifizieren welche Parameter gespeichert werden sollen und die Tracevariable X muss gleich 0 sein.

Zum Beispiel; angenommen die Ist- und die Sollposition der Achse 3 sollen während eines jeden Zyklus gespeichert werden. Die folgenden Kommandos finden Anwendung um die Tracevariablen zu konfigurieren:

Kommando	Beschreibung
SetTraceVariable 0, 0203h	setzt die Tracevariable 0 Parameter 2 (Sollposition) für die 3. Achse zu speichern.
SetTraceVariable 1, 0502h	setzt die Tracevariable 1 Parameter 5 (Istposition) für die 3. Achse zu

Kommando	Beschreibung
	speichern.
SetTraceVariable 2, 0000h	deaktiviere die Tracevariable 2 und darüber.

Trace Modi

Während der Erfassung der Daten werden diese sequentiell im Tracebuffer gespeichert. Wenn das Ende des Tracebuffers erreicht worden ist, wird die Erfassung auf zweierlei Arten reagieren, je nachdem welcher Modus selektiert wurde.

Falls "einmal" selektiert wurde, wird die Erfassung stoppen wenn der Buffer voll ist.

Falls der "kontinuierliche Modus" selektiert wurde, wird die Erfassung wieder am Anfang des Buffers Daten speichern (wrap-around). In diesem Modus wird die diagnostische Erfassung nicht enden ehe die Bedingungen, wie mit Kommando SetTraceStop gesetzt, erfüllt werden.

Verwenden Sie das Kommando SetTraceMode um den Modus auszuwählen. Das Kommando GetTraceMode liefert den programmierten Modus zurück.

Trace Start/Stop Konditionen

Das Kommando SetTraceStart wird verwendet um die Bedingungen zu setzen die den Erfassungsmechanismus in Gang setzen. Ein ähnliches Kommando (SetTraceStop) wird verwendet um die Bedingungen zu setzen welche den Erfassungsmechanismus stoppen. Beide, SetTraceStart und SetTraceStop, sind ein 16-bit Datenwort welches 4 codierte Parameter enthält:

Bits	Name	Beschreibung
0-3	Trigger Axis	Für Triggertypen anders als unmittelbar, bestimmt dieses Feld, welche Achse als die Quelle für den Trigger benutzt werden soll. Verwenden Sie 0 für Achse 1, 1 für Achse 2, usw.
4-7	Trigger Type	Definiert die Art von Trigger die benutzt werden soll. Sehen Sie in der Tabelle unten für eine vollständige Auflistung von Triggertypen nach.
8-11	Bit Number	Für Triggertypen die auf den Statusregister basieren, bestimmt dieses Feld welches Bit (0-15) des Statusregisters überwacht werden soll.
12-15	Bit State	Für Triggertypen die auf den Statusregister basieren, bestimmt dieses Feld welcher Status (0 oder 1) des spezifizierten Bits einen Trigger veranlassen wird.

Das Feld für den Triggertyp muss eines der folgenden Werte beinhalten:

ID	Name	Beschreibung
0	Immediate	Dieser Triggertyp zeigt an, dass die Erfassung sofort startet/stoppt wenn als Typ das Kommando SetTraceStart/SetTraceStop verwendet wurde. Wenn dieser Trippertyp spezifiziert wurde, dann werden die Werte für Quellachse, Bitnummer und Bitstatus nicht verwendet.
1	Update	Die Erfassung startet/stoppt beim nächsten Update der spezifizierten Quellachse. Dieser Triggertyp verwendet nicht die Bitnummer oder Statuswerte.
2	Event Status	Der spezifizierte Bit im "Event Status Register" wird kontinuierlich überwacht. Wenn der Bit den definierten Zustand einnimmt (0 oder 1) startet (stoppt) die Erfassung.
3	Activity Status	Der spezifizierte Bit im "Activity Status Register" wird kontinuierlich überwacht. Wenn der Bit den definierten Zustand einnimmt (0 oder 1) startet (stoppt) die Erfassung.
4	Signal Status	Der spezifizierte Bit im "Signal Status Register" wird kontinuierlich überwacht. Wenn der Bit den definierten Zustand einnimmt (0 oder 1) startet (stoppt) die Erfassung.
5	Drive Status	Der spezifizierte Bit im "Drive Status Register" wird kontinuierlich überwacht. Wenn der Bit den definierten Zustand einnimmt (0 oder 1) startet (stoppt) die Erfassung. (Nur verfügbar auf POSYS® 1800/1900 mit Firmwareversion >2.x)

Download von Trace Daten

Wenn die Erfassung einmal gelaufen ist und der Tracebuffer voll (oder teilvoll) mit Daten ist, können diese Daten, durch Verwendung der Standardkommandos um den Buffer zu lesen, durch den Host heruntergeladen werden.

Wenn eine Erfassung stoppt (entweder durch Setzen des entsprechenden SetTraceStop-Kommandos oder weil das Ende des Tracebuffers erreicht wurde) wird der Lesezeiger des Tracebuffer's auf den ältesten Wert gesetzt. Falls kein Wrap-around stattfand, dann wird dies die Adresse 0 sein. Falls ein Wrap-around stattfand, wird der Lesezeiger auf den Speicherbereich gesetzt der von der nächsten Erfassung mit Daten überschrieben wurde. Zu jeder Zeit kann das Kommando GetTraceCount verwendet werden um die 32-bit Datenwörter, die im Speicher gespeichert wurden, zu lesen.

Dieser Wert kann verwendet werden um die Anzahl der ReadBuffer Kommandos zu bestimmen um den kompletten Inhalt des Tracebuffer's herunter zu laden. Weil der Lesezeiger automatisch auf das älteste Datenwort gesetzt wird wenn die Erfassung endet und weil der Lesezeiger automatisch hochzählt und umbricht während Daten gelesen werden, ist das Lesen des kompletten Inhaltes des Tracebuffer's so einfach wie das zuweisen von x-ReadBuffer Kommandos (wobei x der Wert der zurückgeliefert wird vom Kommando GetTraceCount). Während jeder "Trace Period" wird jede "Trace Variable" benutzt einen 32-Bit Wert im Tracebuffer zu speichern. Daher, wenn Daten aus dem Buffer gelesen werden, wird der 1. Wert der gelesen wird der Variablen 1 zugeordnet, der 2. Wert der Variablen 2, usw.

Beides, die Länge des Tracebuffers und die Anzahl der "Trace Variables" hat direkten Einfluss auf die Anzahl der zu erfassenden Anzahl Daten, die gespeichert werden können. z.B. wenn der Tracebuffer auf 1000 Datenwörter (32-Bit) begrenzt wurde und 2 Variablen initialisiert sind (Variable 0 und 1), dann können bis zu 500 Datensätze gespeichert werden. Jedoch wenn 3 Variablen (0, 1 und 2) erfasst werden, können nur 333 Datensätze gespeichert werden. In diesem Fall wird das übrige Datenwort die erste Variable von der 334. Erfassung speichern.

Wenn der Tracemodus kontinuierlich abläuft, dann wird die 334. Erfassung das 1. Datenwort im letzten verfügbaren Speicher des Tracebuffer's speichern, das 2. und 3. Datenwort werden entsprechend an den Plätzen 0 und 1 abgelegt. In diesem Fall werden die beiden ersten Worte der ersten Erfassung mit den 2 letzten Worten der 334. Erfassung überschrieben. Wenn die Erfassung gestoppt wird zeigt der Lesezeiger auf das älteste Datenwort im Buffer. Dieses Datenwort muss aber nicht das 1. Datenwort der Erfassung sein.

Daher wird empfohlen, dass die Länge des Tracebuffers definiert wird als eine multiple gerade Nummer der Anzahl von Tracevariablen die verwendet werden. Dies würde sicher stellen, dass der Leseindex auf das 1. Wort in einem kompletten Erfassungssatz zeigt, egal ob der Tracemodus einmal oder kontinuierlich abläuft. Die einfachste Lösung wäre es sicher zu stellen, dass die Tracebufferlänge eine multiple Gerade des 12-fachen ergibt (weil jede Anzahl von Variablen, 1, 2 , 3 oder 4, durch 12 teilbar ist).

Traces Starten

Hier ist eine Zusammenfassung der Datenerfassungsmöglichkeiten für den Anfang:

- Trace Buffer definieren. Mit den Kommandos SetBufferStart und SetBufferLength werden die Position im On-Board RAM und die Menge des RAM, die für die Erfassung der Daten bereitgestellt werden soll, definiert. Definieren Sie nicht mehr RAM als tatsächlich physikalisch zur Verfügung steht um die Daten aufnehmen zu können. Seien Sie auch vorsichtig nicht mehr als den tatsächlich zur Verfügung stehenden physikalischen Speicher zu definieren.

- Trace Period definieren. Das Kommando SetTracePeriod erlaubt die Intervalle zu definieren in denen Daten erfasst und gesammelt werden sollen.

- Spezifizieren Sie welche Daten gespeichert werden sollen. Das Kommando SetTraceVariable (bis zu 4) wird verwendet um zu definieren welche Daten pro "Trace Period" erfasst werden sollen. Stellen Sie sicher, dass Variablen 0 - (x-1) verwendet werden, wenn X Variablen gespeichert werden sollen und Variable x gleich 0 ist.

- Definieren Sie den "Trace Mode". Falls die Erfassung aufgrund eines Ereignisses stattfinden soll, dann sollte die "einmal"-Erfassung Verwendung finden. Dies würde es erlauben den kompletten Buffer mit Daten zu füllen und kann mit dem Eintreten des Ereignisses beginnen (Kommando SetTraceStart). Als Alternative, falls die Erfassung aufgrund eines Ereignisses stoppen soll (Kommando SetTraceStop), dann sollte die kontinuierliche Erfassung eingestellt sein. Dies würde das System dazu veranlassen kontinuierlich die Daten zu erfassen bis die Stop-Bedingung erfüllt wurde. Bis zu diesem Punkt würden die Daten, die bis zu dem Stop-Ereignis erfasst wurden, im Buffer gespeichert. Vorher erfasste Daten können wieder überschrieben worden sein, falls die Stop-Bedingung nach einem Wrap-around eintritt und der zur Verfügung gestellte Speicherbereich bereits mit neuen Daten beschrieben wurde.

- Stopmethode definieren (falls gewünscht). Falls ein spezielles Ereignis die Erfassung beenden soll, dann sollten sie mit dem Kommando SetTraceStop programmiert werden. Jedoch, wenn die Erfassung angehalten werden soll wenn der Buffer voll ist ("einmal"-Erfassung), dann ist es nicht notwendig eine Stop-Bedingung zu programmieren. Es ist auch erlaubt mit dem Kommando SetTraceStop eine laufende Erfassung sofort zu stoppen.

- Erfassung starten. Das Kommando SetTraceStart kann verwendet werden eine Erfassung sofort beginnen zu lassen (Triggertyp: sofort). Als Alternative kann ein Ereignis spezifiziert werden, welches die Erfassung starten lassen soll.

¤

Tracking Window

Tracking Window

Die POSYS® stellt programmierbare Tracking Windows zur Verfügung, die verwendet werden können das Servoverhalten außerhalb des Zusammenhanges eines Schleppabstandsfehlers zu überwachen. Das "Tracking Window" funktioniert ähnlich wie beim Schleppabstandsfehler, dadurch, dass es eine programmierbare Positionierfehlergrenze gibt, innerhalb derer die Achse verweilen muss.

Allerdings, im Gegensatz zum programmierbaren Schleppabstandsfehler, falls die Achse sich außerhalb des "Tracking Windows" bewegt, wird die Achse nicht gestoppt. Das "Tracking Window" ist nützlich, wenn externe Prozesse von dem Motor abhängen, bestimmte Regelparameter innerhalb eines Bereiches zu ermitteln. Alternativ kann das "Tracking Window" als eine Art Frühwarnung für Positionierprobleme benutzt werden, die sich noch nicht als Schleppabstandsfehler qualifizieren lassen.

Die Größe des "Tracking Windows" (maximaler erlaubter Positionierfehler innerhalb des "Tracking Windows") wird mit dem Kommando SetTrackingWindow gesetzt. Der Befehl GetTrackingWindow liefert den programmierten Wert zurück.

Wenn der Positionierfehler kleiner als oder gleich dem Wert für das "Tracking Window" ist, ist der Bit im Activity Status Register gesetzt. Falls der Positionierfehler den programmierten Wert überschreitet, wird das Bit gelöscht.

¤

SetEventAction Prozess

SetEventAction

nur für POSYS® 1800/1900 mit Motion Control Firmwareversion >2.x

Die POSYS® 1800/1900 unterstützen einen programmierbaren Mechanismus um auf sicherheitsoder leistungsbezogene Bedingungen zu reagieren.

Das Kommando SetEventAction definiert die auszuführende Aktion bei einer bestimmten Bedingung. Um eine ereignis-bedingte Antwort zu definieren, müssen sowohl die Bedingung als auch die Aktion spezifiziert werden. Die folgende Tabelle listet die möglichen Event Status Register Bedingungen auf die verwendet werden könnenum eine ereignis-bedingte Aktion zu definieren.

Bedingung	Beschreibung
Motion Error	Ein Bewegungsfehler tritt ein wenn ein programmierter Schleppabstandsfehler überschritten wird.
Positive Limit	Ein positives Endschalterereignis tritt ein sobald das korrespondierende Signal aktiv wird solange der Motor eine positive Geschwindigkeit aufweist.
Negaitve Limit	Ein negatives Endschalterereignis tritt ein sobald das korrespondierende Signal aktiv wird solange der Motor eine negative Geschwindigkeit aufweist.

Beides, Motion Error Processing und Endschalterereignisse sind in den entsprecenden Kapiteln näher beschrieben. Zusätzlich zu diesen vier überwachten Bedingungen lassen ist es möglich eine sofortige sicherheits-bedingte Aktion auszuführen.

Die folgende Tabelle beschreibt die ausführbaren Aktionen die für diese Bedingungen programmiert werden können.

Action Name	Beschreibung
No Action	Keine Aktion wird ausgeführt
Smooth Stop	Veranlasst einen Stopp mit programmierte Rampe. Die Geschwindigkeit wird aus Null (0) gesetzt nachdem die EventAction augeführt wurde.
Abrupt Stop	Veranlasst einen sofortigen Stopp des Trajektoriegenerators. Die Geschwindigkeit wird aus Null (0) gesetzt nachdem die EventAction augeführt wurde.
Abrupt Stop with Position Clear	Veranlasst einen sofortigen Stopp des Trajektoriegenerators als auch die Löschung des Schleppabstandsfehlers (= ClearPositionError). Die Geschwindigkeit wird aus Null (0) gesetzt nachdem die EventAction augeführt wurde.
Disable Position Loop and Higher Modules	Deaktiviert den Trajektoriegenerator und Position Loop Module
Disable Current Loop and Higher Modules	Deaktiviert den Trejektoriegenerator, Position Loop und Current Loop Module
Disable Motor Output & Higher Modules	Deaktiviert den Trajektoriegenerator, Position Loop, Current Loop und Motorausgabemodule

Nachdem die Eventaktion programmiert wurde, wird der Motion Prozessor die spezifizierten Konditionen permanent überwachen und die programmierte Aktion ausführen sobald der Event eintritt. Desweiteren werden verwandte Aktionen ausgeführt, wie z.B. das Setzen des entsprechenden Bits im Event Status Register

Um eine EventAction aufzuheben wird das Kommando RestoreOperatingMode verwendet. Dieses Kommando setzt den Motion Prozessor in den Operating Mode zurück der vorher mit dem Kommando SetOperatingMode definiert wurde. Bitte beachten Sie, dass falls die EventAction immer noch aktuell ist, wird sie sofort wieder ausgeführt.

Es liegt in der Verantwortung des Anwenders die Bedingung des ereignis-bedingten Ev ents zu eruieren und auf zu lösen, ehe wieder mit RestoreOperatingMode der Motor wieder in Betrieb genommen werden kann.

Falls als EventAction "No Action", "Abrupt Stop" oder "Smooth Stop" programmiert wurde, wird das Kommando RestoreOperatingMode keinen Einfluss auf die Wiederherstellung des Betriebes haben. Es hat nur Einfluss auf die Wiederherstelluing der Module allein.

Wenn erst programmiert wird ein EventAction aktiv sein bis es reprogrammiert wurde. Das Auftreten einer Ereignisbedingung setzt die definierte EventAction nicht zurück, sie bleibt aktiv.

Die POSYS® 1800/1900 Karten mit Firmware Version >2.x stellt Standardwerte für die ereignis-bedingte Verarbeitung zur Verfügung. Diese Standardwerte sind gedacht einen sicheren Betrieb für viele typische Motion C ontrol Anwendungen bereit zu stellen. Ob diese Standardwerte der Anwendung genügen muss der Anwender bestimmen.

Die Standardwerte lauten wie folgt:

Bedingung	Standardaktion
Motion Error	Deaktiviere Position Loop und Trajektoriegenerator
Positiver und Negativer Endschaltereereignis	Abrupter Stopp mit Löschung des Schleppabstandfehlerwertes.

¤

Hardware Signale

Hardware Signale; Überblick

Hardware Signale; Überblick

Es gibt eine Anzahl Signale für jede Achse der POSYS® die verwendet werden können, um die Aktivität der POSYS® mit Ereignissen von Ausserhalb zu koordinieren. In diesem Abschnitt werden wir diese Signale beschreiben. Dies sind die bi-direktionalen Endschalter, der AxisIn-Eingang und der AxisOut-Ausgang. Diese Signale existieren für jede Achse der POSYS®.

GetAxis_In_Out_Home

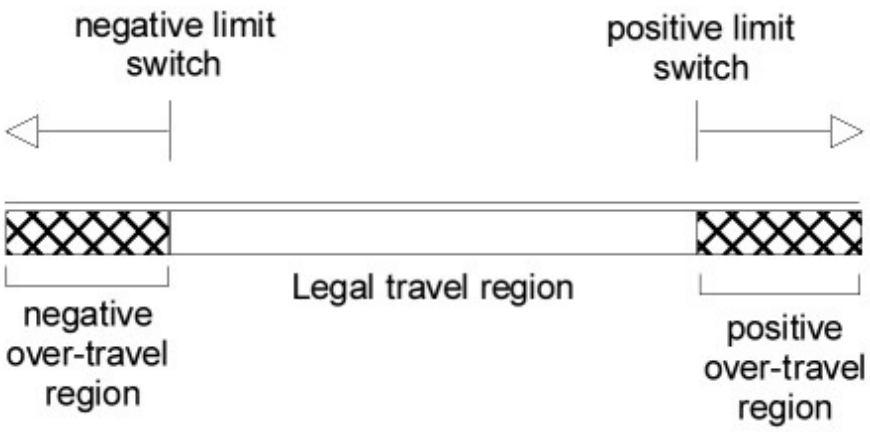
¤

Endschalter

Endschalter

Die POSYS® unterstützt Endschalterereignisse die verwendet werden können, um automatisch eine "Bewegungsende"-Bedingung zu erkennen. Dies ist ein wichtiges Sicherheitsmerkmal für Positioniersysteme, die eine definierte Reichweite haben.

Die positiven und negativen Endschalter werden mit POSYS®-Eingängen Poslim1-4 beziehungsweise Neglim1-4 verbunden, um das Anschlagen am positiven bzw. negativen Endschaltereingang zu entdecken.



Richtungsabhängige Endschalteroperation

Es gibt zwei Hauptfunktionen, die die POSYS® in Verbindung mit den Endschaltereingängen unterstützt:

- Der Host kann automatisch mit der Information versorgt werden, wenn ein Endschalterereignis eintritt, um den Host in die Lage zu versetzen, entsprechende Aktionen einzuleiten.
- Indem eine Endschalterbedingung eingegeben wird, kann die Bewegung zu einem Stopp geführt werden, damit der Motor nicht weiter in die "Verbotene Zone" weiter fährt.

Die Bearbeitung von Endschalterereignissen kann mit dem Kommando SetLimitSwitchMode aktiviert oder deaktiviert werden. Der selbe Register kann mit dem Kommando GetLimitSwitchMode gelesen werden.

Falls die Endschalterüberwachung der POSYS® eingeschaltet wurde, werden die Eingänge kontinuierlich auf ein Endschalterereignis hin überwacht. Ein Endschalterereignis entsteht dann wenn ein Endschaltereingang aktiv wird, wenn sich die Achse in Richtung der Sollposition des Endschalters bewegt. Falls sich die Achse nicht bewegt, in Open-Loop-Mode befindet oder sich in die entgegengesetzte Richtung bewegt, wird kein Endschalterereignis eintreten. Z.B. wird ein Endschalterereignis dann eintreten, wenn die Sollposition sich in positiver Richtung befindet und der Endschalter für die positive Richtung auf "high" schaltet. Jedoch wird dies nicht eintreten wenn die Sollposition sich in negativer Richtung

befindet oder die Achse sich nicht bewegt.

Der Zustand der Endschaltereingänge (active high oder active low) kann mit dem Kommando SetSignalSense definiert werden.

Wenn ein Endschalterereignis eintritt, generiert die POSYS® einen abrupten Stopp. Zusätzlich wird der korrespondierende Bit im "Event Status Register" gesetzt. Und schließlich wird auch der korrespondierende Bit im "Activity Status Register" gesetzt.

Nachdem ein Endschalterereignis eingetreten ist, sollten folgende Schritte unternommen werden, um das Ereignis zu klären:

- Außer wenn Endschalterereignisse während normaler Maschinenoperation vorkommen können, sollte die Ursache des Ereignisses untersucht werden und entsprechende Sicherheitskorrekturen gemacht werden.
- Der/die Endschalterbit(s) im "Event Status Register" sollte(n) durch das Kommando ResetEventStatus gelöscht werden. Bewegung in einer Richtung ist nicht möglich (nämlich jene in der der Motor auf den Endschalter aufgefahren ist), während ein Endschalterbit im "Event Status Register" gesetzt ist.
- Eine Bewegung in die entgegengesetzte Richtung sollte initiiert werden, welche das Endschalterereignis ausgelöst hat. Dies kann jede Art von Bewegung sein die die Achse in die andere Richtung vom Endschalter herunterführt. Sollte aber die Achse den Versuch unternehmen weiter in die Richtung auf den Endschalter zu fahren, wird ein neues Endschalterereignis und ein Kommando-Fehler ausgelöst.

Falls ein Endschalter sowohl mit dem positiven als auch negativen Eingang der POSYS® verkabelt sind, ist es nicht mehr möglich weder in die eine noch in die andere Richtung herunter zu fahren. Jedoch, falls diese Anordnung gewünscht wird (vermutlich aufgrund einer besonderen Verdrahtungsanordnung), werden dann beide Endschalterbits im "Activity Status Register" gesetzt, welches zur Folge hat, daß eine Bewegung weder in die eine noch in die andere Richtung möglich ist. In diesem Fall sollte das Kommando SetLimitSwitchMode verwendet werden um die Endschalterüberwachung zu deaktivieren, während der Motor vom Endschalter heruntergefahren wird.

Anmerkung: Endschalter funktionieren nicht wenn die Achse im Modus "Motor OFF" ist, auch bekannt als Open-Loop-Mode.

¤

AxisOut Output

Der AxisOut Pin

Jede Achse hat einen für allgemeine Zwecke ausgelegten Ausgang, welches programmiert werden kann auf jeden Zustand der zugewiesenen Bits im "Event Status Register, "Activity Status Register" oder "Signal Status Register" zu agieren. Dieses überwachte Bit in einem der drei Register kann der gleichen Achse oder anderen Achse als die Achse des AxisOut-Ausganges selbst sein. Diese Funktion ist nützlich um Hardwaresignale auszugeben um externe Peripheriegeräte zu triggern.

Das Kommando Set/GetAxisOutSource kann verwendet werden um den AxisOut-Ausgang zu konfigurieren. Das Kommando benutzt ein einzelnes Wort als Argument. Der Wert dieses Parameters wird folgendermaßen interpretiert:

Bits	Name	Beschreibung
0-3	Source axis	Spezifiziert die Achse die als Quelle für das Achsenausgangssignal verwendet werden soll. Der Achsenausgangspin wird dem angegebenen Registerbit der Quellachse folgen. Eine 0 identifiziert Achse 1, eine 1 die Achse 2 usw.
4-7	Bit number	Zeigt an welches Bit in dem ausgewählten Zustandsregister vom Achsenausgangspin gefolgt wird. Bits werden von 0-15 durchnummeriert, wobe Bit 0 das niederwertigste Bit anzeigt.
8-11	Status Register	Zeigt an welches Register als die Quelle benutzt wird für den Achsenausgang. D Kodierung ist folgendermaßen: ID — Register 0 — Keine, der Achsenausgang ist immer dektiviert 1 — Event Status Register 2 — Activity Status Register 3 — Signal Status Register 4-15 — reserviert, nicht verwenden
12-15	Reserviert	Reserviert für zukünftige Verwendung. Sollten Nullen sein.

Anmerkung: der Achsenausgang kann „active high“ oder „active low“ durch das Kommando SetSignalSense programmiert werden.

Es ist möglich den AxisOut-Ausgang als ein direktes unter Hostkontrolle laufendes Software-programmiertes Ausgangsbit zu benutzen. Dies kann bewerkstelligt werden indem man null selektiert als Register-ID mit dem Kommando SetAxisOutSource und indem man den inaktiven Ausgangsstatus mit dem Kommando SetSignalSense als „active high“ oder „active low“ je nach Bedarf einstellt.

¤

AxisIn Input

Der AxisIn Pin

Jede Achse hat einen für allgemeine Zwecke verwendbaren achsenspezifischen Eingang, welches mit dem Kommando GetSignalStatus gelesen werden kann und um auch automatisch Ereignisse zu erfassen, wie z.B. durch einen Signalübergang erzeugte Bewegungsänderung (Stop, Start, Geschwindigkeitsänderung usw.) aufgrund der Verwendung von Breakpoints.

Es sind keine weiteren speziellen Kommandos notwendig um AxisIn zu aktivieren oder deaktivieren.

¤

Analog Eingang

Analog Eingang

Die POSYS® verfügt über 8, für allgemeine Zwecke verwendbare analoge Eingänge. Diese 8 Eingänge sind mit einer internen Schaltung verbunden, die die analogen Signale in digitale Signale mit 10 Bit Auflösung umwandelt. Die Konvertierung geschieht kontinuierlich und passiert auf allen 8 Kanälen alle 4 Zyklen. Für eine 4-Achsen POSYS® 18xx/19xx geschieht dies alle 1024 µs. Für alle 4-Achsen POSYS® der Baureihe 7xx/8xx/8xx-B/9xx geschieht dies alle 1640 µs/2480 µs, je nach Modell.

Mit dem Kommando ReadAnalog wird der zuletzt konvertierte Wert gelesen. Der Wert der mit diesem Kommando zurückgeliefert wird, resultiert daraus indem der konvertierte 10-Bit Wert 6 Bit nach rechts versetzt wird. Die Analogdaten sind für allgemeine Zwecke gedacht und werden nicht vom Motionprozessor für irgendwelche Kalkulationen verwendet.

Diese analogen Eingänge sind geeignet die POSYS® mit Joysticks zu verbinden um z.B. eine Achse per Handbetrieb in eine bestimmte Position zu bringen, oder für Teaching-Funktionen. Hierbei können in Verbindung mit der Trace-Funktion die Positionen abgespeichert und anschliessend vom Host ausgelesen werden. Damit wären komplette, aufwendige Konturen erfass- und anschliessend abfahrbar.

Welche Spannungen verwendet werden dürfen entnehmen Sie bitte den POSYS® Versionsspezifikationen.

¤

Der SYNC IO - Multiple Controller Synchronization

Der Sync IO - Multiple Controller Synchronization

Die POSYS® unterstützen die Synchronsation der internen Zykluszeit mehrerer Controller. Dies erlaubt die Synchronisation von Start/Stop und die Modifizierung von Bewegungsabläufen über mehrere Controller hinweg wo präzises Timing erforderlich ist. Dies kann notwendig sein, weil die Regler sich in größerer Entfernung befinden oder die Applikation mehr als 4 Achsen von synchronisierter Bewegung erfordern. In der üblichsten Konfiguration wird ein Regler als ein Master ausgewiesen und alle anderen Regler werden in den Slave-Modus gesetzt.

Multi-Karten-Synchronisation ist nicht möglich sobald eine Achse im Schrittmotormodus betrieben wird.

Der Eingang/Ausgangs Status von diesem IO und seiner Aufgabenstellung werden mit dem Kommando

SetSynchronizationMode gesetzt.

Die Tabelle fasst unten die Modi zusammen.

Encoding	Modus	Beschreibung
0	Disabled	In diesem deaktivierten Modus ist die Verbindung als Eingang konfiguriert und nicht in Verwendung. Dies ist der Standardmodus nach einem Reset oder nach dem Einschalten.
1	Master	Im Master-Modus gibt der Controller einen Synchronisationstakt aus der von den Slave-Controllern oder anderen Geräten verwendet werden kann um den internen Arbeitszyklus mit dem Master zu synchronizieren.
2	Slave	Im Slave-Modus ist der Controller derart konfiguriert dass das eingehende Synchronisationssignal den internen Arbeitszyklus synchronisiert mit dem Eingangssignal.

Wenn man mehrere Regler synchronisiert, müssen die folgenden Regeln beachtet werden:

- Die Servoloopupdaterate aller Regler muss gleich sein. Z.B. wenn eine 4-Achsen-POSYS® und eine 2-Achsen POSYS® miteinander synchronisiert werden sollen, muss mindestens der Größere der beiden möglichen niedrigsten Werte eingestellt sein.
- Nur eine Karte im Verbund kann als Master fungieren. Z.B. wenn 3 Controller zu einem Verbund zusammen geschaltet wurden, kann nur eine Karte als Master, die anderen beiden Karten müssen als Slaves konfiguriert sein.
- Im Verbund müssen zuerst die Slaves definiert werden und dann die letzte Karte als Master. Dies stellt sicher, dass die Zykluszeiten der Slaves präzise auf die Zykluszeit des Master eingestellt werden.
- Es können nur Karten der Reihen 8xx-B und 9xx bzw. 19xx und 18xx mit- und untereinander synchronisiert werden.

Der Master sendet alle 50 µs ein Taktsignal an die Slaves. Die Applikationen des Anwenders können dieses Signal auch zur Synchronisation weiterer externer Peripheriegeräte verwenden.

¤

Motoranbindung

Überblick

Überblick

Die POSYS® unterstützt 2 verschiedene Arten der Positionsrückmeldung: imkrementale Enkoderrückmeldung und Parallelwortrückmeldung (Resolver und Absolutwertgeber). Die Art der Rückmeldung ist programmierbar, aber die notwendige Hardware muss installiert sein damit die gewählte Variante verwendet werden kann (Parallelwortrückmeldung ist optional).

Um die Art der Enkoderrückmeldung zu wählen wird das Kommando SetEncoderSource verwendet. Um die gewählte Einstellung abrufen zu können verwendet man das Kommando GetEncoderSource.

¤

Inkrementaler Enkodereingang

Inkrementaler Enkodereingang

Die inkrementale Rückmeldung liefert 2 Rechtecksignale zurück. A und B und einen optionalen Index, der normalerweise anzeigt wann der Motor eine volle Umdrehung zurückgelegt hat. Das Offset der beiden Signale A und B beträgt 90°.

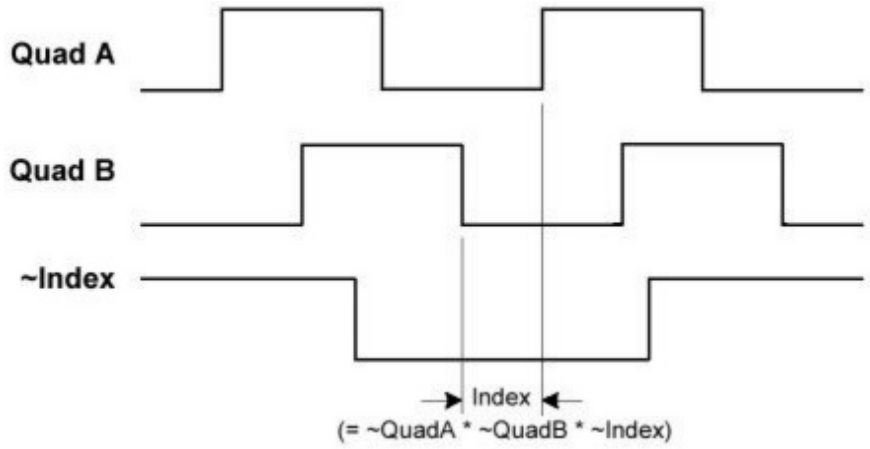


Figure 8-3. Quadrature Encoder Timing

Damit die Enkoderposition von der POSYS® korrekt interpretiert werden kann, sollte das A-Signal dem B-Signal vorangehen wenn der Motor sich in positiver Richtung bewegt (Figur 8-3). Wenn der Motor sich in negativer Richtung dreht, soll das B-Signal dem A-Signal vorausgehen. Wegen des 90° Offsets erhält die POSYS® 4 Takte für eine voll erfasste Phase für beide Signale, A und B.

Die POSYS® unterstützt auch die Verwendung invertierter Signale, was eine wesentlich höhere Signalsicherheit gewährleistet. Hierzu müssen für beste Signalqualität die Netzwiderstände (RS1, RS2 und RS3) gesteckt sein.

¤

Aktuelle Positionsregister

Aktuelle Positionsregister

Die POSYS® überwacht kontinuierlich die Rückmeldung und akkumuliert einen 32-Bit Positionswert, genannt "Actual Position". Nach dem Einschalten ist die aktuelle Position gleich null. Die akuelle Position kann allerdings mit dem

Kommando SetActualPosition neu definiert werden und mit dem Kommando GetActualPosition kann die aktuelle Position abgefragt werden.

Im Allgemeinen bei Verwendung von Inkrementalgebern, wird die aktuelle Position kurz nach dem Einschalten gesetzt, indem eine Prozedur aufgerufen wird mit deren Hilfe eine physikalisch definierte Hardwareposition als Referenz angefahren wird.

¤

Digitale Filterung

Digitale Filterung

Alle Enkoder- als auch der Index- und die Homeinputs sind digital gefiltert um die Verlässlichkeit zu erhöhen. Der Filter erfordert, dass ein gültiger Übergang nur dann akzeptiert wird, wenn es den neuen Status (low oder high) für zumindest 3 Zyklen á 50 ns (Gesamt 150 ns) beibehält. Dies soll sicherstellen, dass kurze Störsignale nicht als Enkodersignale fehlinterpretiert werden.

Obwohl dieses digitale Filterungsschema die allgemeine Verlässlichkeit erhöht kann es trotzdem notwendig sein zusätzliche Techniken einzusetzen um die höchstmögliche Verlässlichkeit zu erreichen, wie z.B. der Einsatz von Differential Line Drivers/Receivers, oder analoge Filterung. Ob diese zusätzlichen Filterungen notwendig sind, hängt vom System ab und ob und wie viele Störsignale auftreten mögen.

¤

High Speed Position Capture

High Speed Position Capture

Jede POSYS® unterstützt einen High Speed Positionserfassungsregister um die aktuelle Achsenposition zu erfassen (pro Achse), getriggert durch ein externes Signal. Eines von zwei Signalen kann als Eingangstrigger verwendet werden um entweder das Indexsignal oder die Home Position zu triggern.

Diese beiden Eingänge differieren in der Art wie die Erfassung erkannt wird. Falls das Indexsignal getriggert wird, wird das Signal gesetzt wenn das A, B und Indexsignal eine bestimmte Position einnehmen (definiert im Signal Sense Register der mit dem Kommando SignalSenseRegister verwendet wird). Falls die Home Position gewählt wurde die es zu erfassen gilt, dann muss nur dieses Signal einen bestimmten Zustand einnehmen um getriggert zu werden.

Die voreingestellten Werte für das A, B, Index- und Homezustandssignal im "Signal Sense Register" ist 0, was bedeutet dass diese Signale "active low" sind. In diesem Zustand, wenn Index getriggert werden soll, wird eine Erkennung dann stattfinden, wenn A, B und Index alle "low" sind. Jede Änderung in der Zustandsmaske (Signal Sense Mask) wie das Signal interpretiert werden soll, würde dementsprechend eine Änderung in der Erkennung des Triggerzustandes zur Folge haben.

Das Kommando SetCaptureSource definiert ob das Index- oder das Homesignal für die Positionserfassung getriggert werden soll. Das Kommando GetCaptureSource liefert den programmierten Wert zurück.

Wenn eine Erfassung getriggert wurde, werden die Inhalte der aktuellen Position in das "Position Capture Register" transferiert und der Erfassungserkennungsindikator (Bit 3 des Event Status Register) wird gesetzt. Mit dem Kommando GetCaptureValue kann man den "Capture Register" lesen. Der "Capture Register" muss gelesen werden, ehe eine neue Erfassung stattfinden kann. Das Lesen des "Capture Register" setzt den Trigger zurück, was bedeutet dass eine weitere Erfassung stattfinden kann. Wie für alle Event Status Register Bits kann das Kommando ResetEventStatus verwendet werden um den Positionserfassungsindikator zu löschen.

¤

Parallelwort Positionsrückmeldung

Parallel-Word Position Input (Option)

Für Systeme die keine inkrementale Rückmeldung unterstützen, stattdessen aber ein digitales Binärwort, bietet die

POSYS® eine optionale Steckerverbindung, welches verwendet werden kann, um eine Vielzahl unterschiedliche Rückmeldeeinheiten einzusetzen. Unter anderem Folgende:

- Resolver (nach Konvertierung von Resolver zu Digital)
- Absolute optische Enkoder
- Laser Interferometer mit parallel-Wort Ausgang
- Inkrementalgeber mit externem Quadraturdecodiererstromkreis
- AD Konverter die ein analoges Rückmeldesignal lesen

In diesem Positionseingangsmodus liest die POSYS® ein 16-Bit Datenwort um die Enkoderposition zu erkennen, wählbar für jede Achse. Abhängig von der Art der Rückmeldeeinheiten können geringere Werte als 16-Bit Auflösung verwendet werden, wobei die oberen Datenbits als Nullen an die POSYS® übergeben werden müssen. Es ist auch annehmbar diese Bits zu erweitern, allerdings unter keinen Umständen sollten unbenutzte Bits des Parallelwortes "hängen" gelassen werden.

Der an die POSYS® übergebene Wert sollte binär codiert sein. Die POSYS® nimmt an, dass die Positionsdaten die von der externen Rückmeldeeinheit zurückgeliefert werden ein zweier Kompliment ist, obwohl der zurückgelieferte Wert im Bereich von 0 bis 2^n-1 rangiert (n ist die Anzahl Bits die von der Rückmeldeeinheit zurückgeliefert werden), dann wird die Differenz in der Verhaltensweise die Interpretation des Startpunktes sein, welches um die Hälfte der vollen Skala der Rückmeldeeinheit versetzt wird. Falls gewünscht kann diese Position mit dem Kommando SetActualPosition geändert werden.

¤

Multi-Turn Systeme

Multi-Turn Systeme

Zusätzlich zur Unterstützung von Positionserfassungen über einen vollen numerischen Bereich, unterstützt die POSYS® auch sogenannte Multi-turn Systeme. Die Parallelenkoderwerte die eingelesen werden, werden kontinuierlich überprüft und ein Überlauf automatisch erkannt, sowohl in positiver als auch in negativer Richtung. Die Werte werden hinzu addiert.

Bei Verwendung dieses "virtuellen" Multi-Turn Zählers speichert die POSYS® kontinuierlich die Position in einem 32-Bit Wert. Natürlich, wenn die Rückmeldeeinheit keinen Überlauf produziert (non Multi-Turn System), wird ein 16-Bit Wert zurückgeliefert.

Wenn der Motor sich in positiver Richtung bewegt, erhöht sich der Wert bis zu einem Maximum. An diesem Punkt des Überlaufes wird von 0 aus weitergezählt und wieder kontinuierlich hochgezählt. Das Gleiche geschieht wenn sich der Motor in negativer Richtung bewegt. Der Wert bei dem der Zähler der Rückmeldeeinheit überläuft, wird "Devices's Modulus" genannt und sollte mit dem Kommando SetEncoderModulus gesetzt werden.

Anmerkung: das Kommando SetEncoderModulus verwendet als Parameter einen Wert der die Hälfte des Modulus beträgt.

Z.B. ein Motor verwendet einen 12-Bit Resolver als Rückmeldeeinheit. Der Modulus beträgt 4.096, der Wert der mit SetEncoderModulus daher gesetzt werden soll, beträgt 2.048. Jedesmal wenn der Binärwert seinen höchsten Wert erreicht und wieder auf seinen kleinsten "zurückspringt", erkennt dies die POSYS® und der Wert zu dem der Überlauf passiert, wird zur aktuellen Position hinzu gezählt mit Werten größer als 4.096 oder kleiner als 0.

Für Systeme die einen Positionszähler mit einem Modulus kleiner als die maximale Anzahl Striche/Umdrehung verwenden (Enkoderauflösung pro Umdrehung), muss der Wert für die Anzahl Striche pro Umdrehung gleich dem Wert für die Positionszählergröße sein. Z.B. wenn ein rotierendes Laserinterferometer verwendet wird, welches ein 16-Bit Wert ausgibt, aber 16.777.216 Zähler pro Umdrehung ausgibt, muss der Zähler pro Umdrehungswert 32.768 (2^{16}/2) sein.

Mit den Parallelwortrückmeldeeinheiten ist eine High Speed Positionserfassung nicht möglich. Diese für diesen Zweck gedachten Signale (A, B, Index und Home) werden in diesem Modus nicht verwendet. Nichsdestotrotz ist die Erfassung über diese Schnittstelle sehr schnell und zuverlässig. Mit unserem SSI800/900 für Absolutwertgeber mit SSI-Schnittstelle wird jede Achse bis zu 20.000x pro Sekunde gelesen.

Trotzdem können für nichtpositionäre Erfassungsfunktionen, die den "Signal Status Register" verwenden, diese Signale normal verwendet werden. Z.B. können diese Bits gelesen (Kommando GetSignalStatus) oder zum Triggern von Unterbrechungspunkten verwendet werden etc.

¤

Parallel-word Device Interfacing

Für jede Achse die auf Parallelwortrückmeldung eingestellt wurde, verwendet die POSYS® den Peripheriebus und die Adressen die nachfolgend beschrieben sind um die Werte der jeweiligen Achse zu lesen:

Peripheriebusadresse	POSYS® I/O Operation	Addressierte Achse #
800h	16-bit peripheral read	1
801h	16-bit peripheral read	2
802h	16-bit peripheral read	3
803h	16-bit peripheral read	4

Für jede Achse die eingestellt wurde Parallelwortdaten einzulesen, wird die POSYS® an der entsprechenden Adresse die Daten auswerten, Achsen die dagegen nicht für diesen Modus programmiert wurden, werden weiterhin die TTL-Eingänge für die Auswertung von Daten der Rückmeldeeinheiten verwenden.

Für eigene Entwicklungen die an diese Schnittstelle angebunden werden sollen, sind nachfolgende Informationen von Bedeutung. Das folgende zeigt die Pinouts für den Option Connector.

Für zusätzliche Informationen über die Anbindung an diese Schnittstelle (z.B falls es in Ihrem Interesse liegt eine eigene Lösung für diese Schnittstelle zu entwickeln; Speichererweiterung, zusätzliche digitale oder analoge E/As usw.), kontaktieren Sie bitte POSYS Motion Control GmbH & Co.KG.

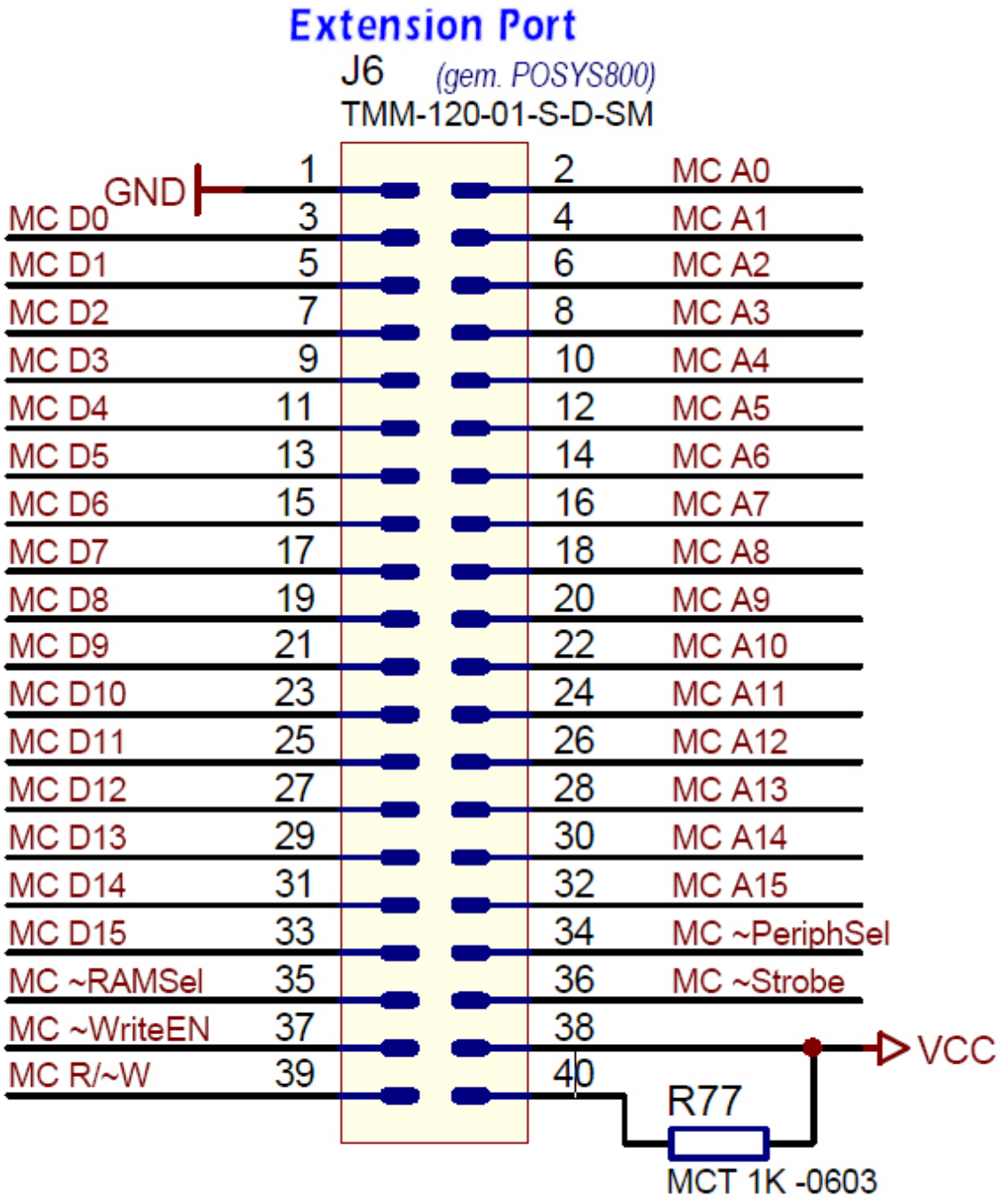
¤

Parallel Device I/O

Der Motion Control Chipsatz verwendet den externen Peripheriebus um Schreib-/Lesevorgänge mit den externen Periperiegeräten des Anwenders zu ermöglichen sofern sie mit bestimmten zugewiesenen C hipsatzfunktionen assoziiert sind, wie z.B. DAC (Digital to Analog Converter) Signalausgabe, Parallelworteingabe, benutzerdefinierten Speicher IO und serielle Schnittstellenkonfigurationseingang.

Der Bus stellt 5 Kontrollsignale zur Verfügung; ~PeriphSlct, ~Strobe, W/~R, R/~W, and ~WriteEnbl. Zusätzlich exisitieren 16 Adresslinien (Addr0-Addr15) und 16 Datenlinien (Data0-Data15). Der Motion C ontroller manipuliert/verwendet diese Kontroll-, Adress- und Datenlinien dahingehend, dass es dem Anwender möglich gemacht wird, eigene externe Peripheriekomponenten zu entwickeln und direkt in den Motion C ontrol Chipsatz zu schreiben bzw. von ihm zu lesen.

Das nachfolgende Diagramm gibt Aufschluss über die Pinbelegung dieser Schnittstelle. Als Beispiel für die Verwendung dieser Schnittstelle dient auch unsere SSI800/900 Karte mit der es dem Anwender ermöglicht SSI-kompatible Absolutwertgeber anzuschliessen.


Extension Port
J6
(gem. POSYS800)
TMM-120-01-S-D-SM
GND
1
2
MC A0
MC D0
3
4
MC A1
MC D1
5
6
MC A2
MC D2
7
8
MC A3
MC D3
9
10
MC A4
MC D4
11
12
MC A5
MC D5
13
14
MC A6
MC D6
15
16
MC A7
MC D7
17
18
MC A8
MC D8
19
20
MC A9
MC D9
21
22
MC A10
MC D10
23
24
MC A11
MC D11
25
26
MC A12
MC D12
27
28
MC A13
MC D13
29
30
MC A14
MC D14
31
32
MC A15
MC D15
33
34
MC ~PeriphSel
MC ~RAMSel
35
36
MC ~Strobe
MC ~WriteEN
37
38
VCC
MC R/~W
39
40
R77
MCT 1K -0603

¤

Peripheral Device Read

Um einen Lesevorgang am Peripheriebus zu ermöglichen wird der C hipsatz das ~PeriphSlct Signal, das ~Strobe signal und das W/~R Signal auf low setzen. Zusätzlich wird der C hipsatz eine oder mehrere Adressleitungen auf high setzen, abhängig von der Anzahl Adressen die es gilt zu decodieren für eine bestimmte Funktion. Nachdem diese Signale gesetzt wurden, wird es erwartet, dass des Benutzer's Peripherieschaltung die korrekten Daten an den Datenbus0-15 übergibt.

Das "~"-Zeichen zeigt an das ein Signal low ist.

Die folgende Tabelle zeigt die Summe der verschiedenen Peripherieadressen für den Navigator und Magellan C hipsatz:

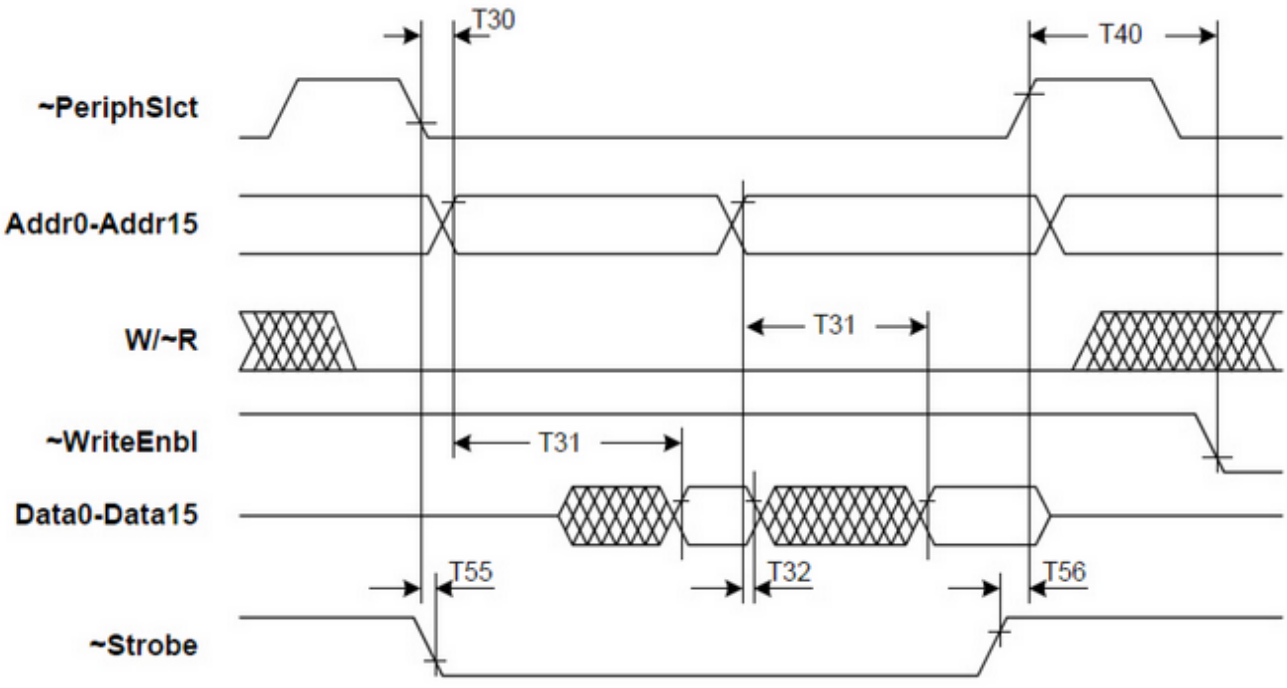
Basisadresse	zu decodierende Adressenbits	Device	Beschreibung
0800h - 0803h	A0, A1, A11	Parallel-word encoder	Adressen für Parallelwort-rückmeldeeinheiten für die Achsen 1-4
1000h - 10ffh	A0-A7, A12	User-defined	Adressen für benutzerdefinierte E/A Einheiten.

Das spezifische Timing für die Peripherielesefunktion ist:

POSYS® 700/800/800-B/900

Timing Interval	*Tn*	Minimum	Maximum
Address Setup Delay Time	T30		7 nsec
Data Access Time	T31		71 nsec
Data Hold Time	T32		2 nsec
Read to Write Delay Time	T40	50 nsec	
PeriphSlct Low to Strobe Low	T55		1 nsec
PeriphSlct High to Strobe High	T56		4 nsec

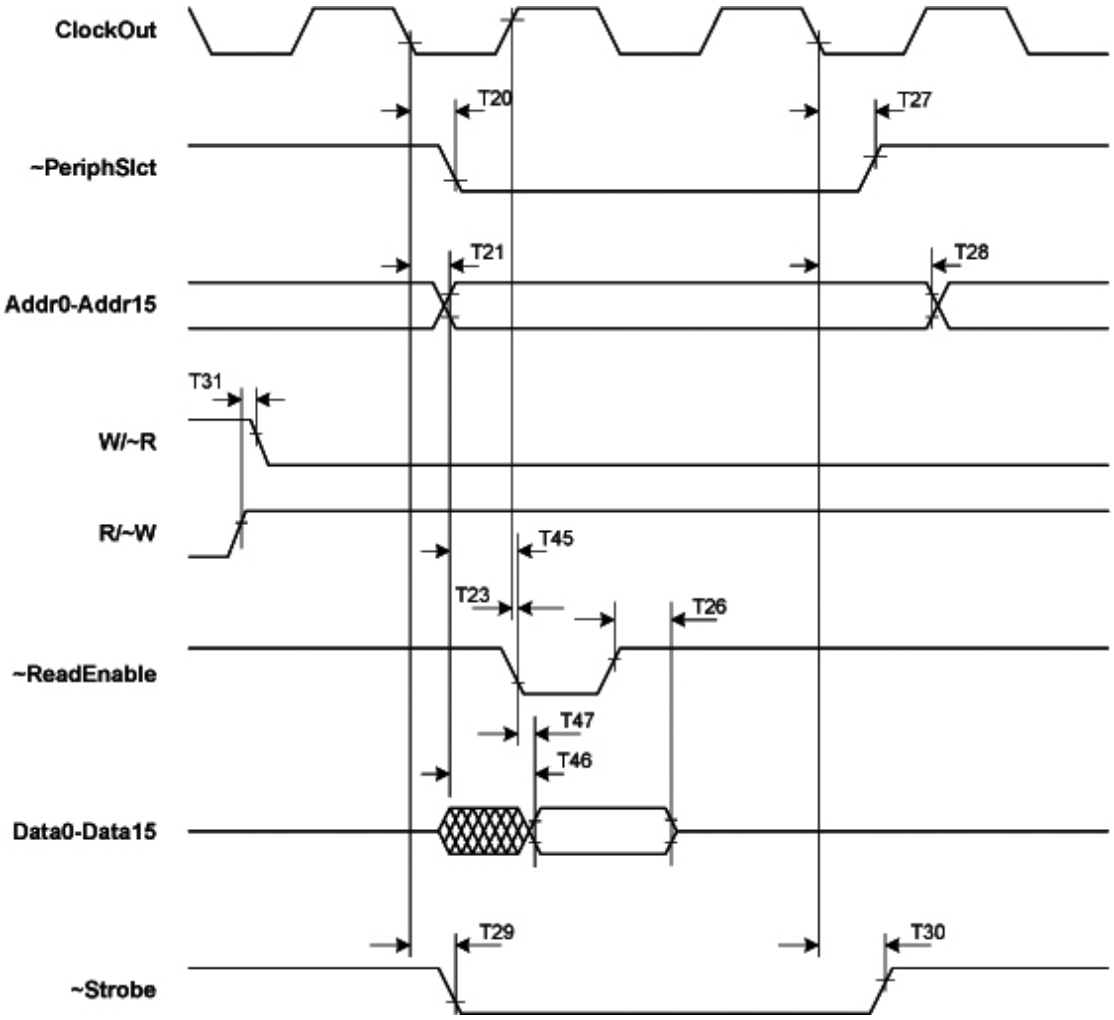
Note 1 Leistungsfiguren und Timinginformationen nur gültig @ Fclk = 40.0 MHz.
Note 2 Nur für 8/8 und 8/16 Interface Modes.
Note 3 Der Clock low/high Split hat einen erlaubten Bereich von 45-55%.



®

POSYS® 1800/1900

Timing Interval	*Tn*	Minimum	Maximum
External Memory Read			
ClockOut low to control valid	T20		4nsec
ClockOut low to address valid	T21		8nsec
Address valid to ~ReadEnable low	T22	31nsec	
ClockOut high to ~ReadEnable low	T23		5nsec
Data access time from Address valid	T24		40nsec
Data access time from ~ReadEnable low	T25		31nsec
Data hold time	T26	0nsec	
ClockOut low to control inactive	T27		5nsec
Address hold time after ClockOut low	T28	2nsec	
ClockOut low to Strobe low	T29		5nsec
ClockOut low to Strobe high	T30		6nsec
W/~R low to R/~W rising delay time	T31		5nsec
Peripheral Device Read			
Address valid to ~ReadEnable low	T45	56nsec	
Data access time from Address valid	T46		65nsec
Data access time from ~ReadEnable low	T47		56nsec



¤

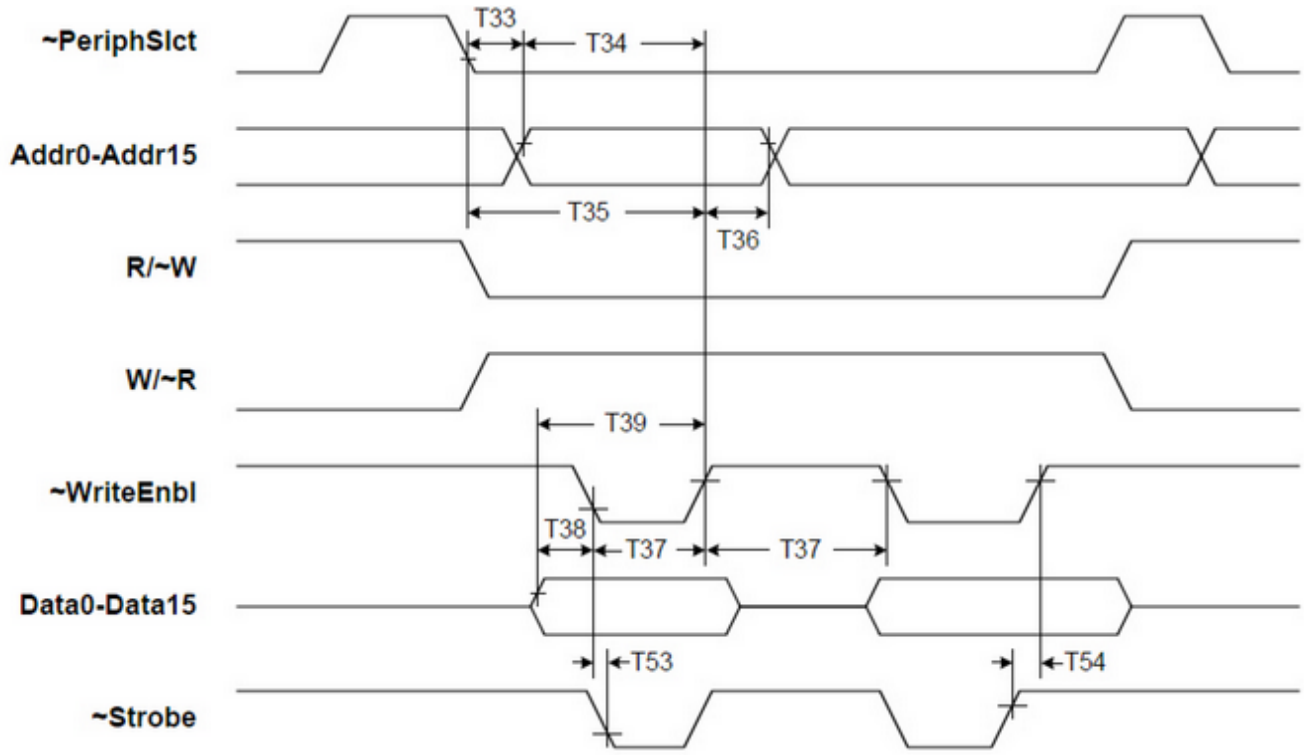
Peripheral Device Write

Um einen Schreibvorgang am Peripheriebus zu ermöglichen wird der C hipsatz das ~PeriphSlct Signal, das R/~W Signal und das ~WriteEnable Signal auf low setzen. Zusätzlich wird der C hipsatz eine oder mehrere Adressleitungen auf high setzen, abhängig von der Anzahl Adressen die es gilt zu decodieren für eine bestimmte Funktion, und zusätzlich noch das Datenwort das durch den C hipsatz geschrieben wird.

Das Symbol "~" zeigt an dass ein Signal low ist.

POSYS® 700/800/800-B/900

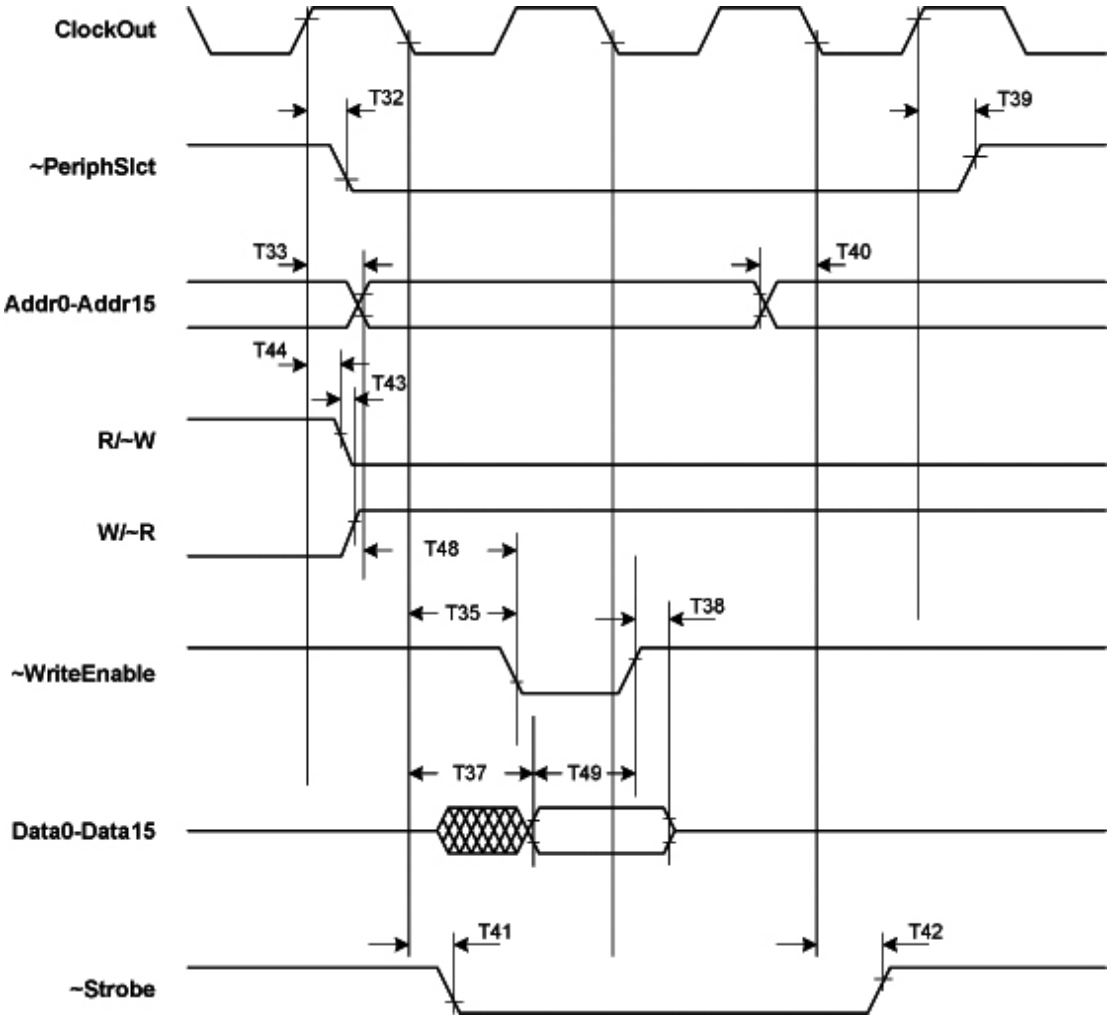
Timing Interval	*Tn*	Minimum	Maximum
Address Setup Delay Time	T33		7 nsec
Address Setup to WriteEnable High	T34	122 nsec	
PeriphSlct Low to WriteEnable High	T35		129 nsec
Address Hold Time	T36	17 nsec	
WriteEnable Pulse Width	T37	89 nsec	
Data Setup Time	T38		3 nsec
Data Setup before Write High Time	T39		92 nsec
WriteEnable Low to Strobe Low	T53		1 nsec
WriteEnable High to Strobe High	T54		3 nsec



POSYS 1800/1900

Timing Interval	*Tn*	Minimum	Maximum
External Memory Write			
ClockOut high to control valid	T32		4nsec
ClockOut high to address valid	T33		10nsec
Address valid to ~WriteEnable low	T34	29nsec	
ClockOut low to ~WriteEnable low	T35		6nsec
Data setup time before ~WriteEnable high	T36	33nsec	
Data bus driven from C lockOut low	T37	-3nsec	

Data hold time	T38	2nsec	
ClockOut high to control inactive	T39		5nsec
Address hold time after ClockOut low	T40	-5nsec	
ClockOut low to Strobe low	T41		6nsec
ClockOut low to Strobe high	T42		6nsec
R/~W low to W/~R rising delay time	T43		5nsec
ClockOut high to control valid	T44		6nsec
Peripheral Device Write			
Address valid to ~WriteEnable low	T48	54nsec	
Data setup time before ~WriteEnable high	T49	58nsec	



¤

Motorausgabesignale

Verschiedene Methoden stehen bei der POSYS® zur Verfügung mit entsprechenden Treibern und Verstärkern verbunden und verwendet zu werden, jedoch variieren die Methoden bei den verschiedenen Motorversionen und Modellen.

Motortype	POSYS® Serie	Phasen pro Achse	Verfügbare Ausgabemethoden
DC bürstenbehaftete Servomotoren	POSYS® 70x	1	PWM Sign/Magnitude PWM 50/50

Motortype	POSYS® Serie	Phasen pro Achse	Verfügbare Ausgabemethoden
	80x 80x-B, 81x-B, 87x-B, 88x-E 90x, 91x, 97x, 98x 180x, 182x 190x, 192x		16-Bit DAC
DC bürstenlose Servomotoren mit externe Kommutierung	POSYS® 70x 80x 80x-B, 81x-B, 87x-B, 88x-E 90x, 91x, 97x, 98x 180x, 182x 190x, 192x	1	PWM Sign/Magnitude PWM 50/50 16-Bit DAC
DC bürstenlose Servomotoren mit interne Kommutierung	POSYS® 82x-B, 83x-B, 87x-B, 88x-E 92x, 93x, 97x, 98x 182x 192x	2 oder 3	PWM Sign/Magnitude * PWM 50/50 16-Bit DAC
Mikroschritt mit Sinus/Cosinus Signalausgabe	POSYS® 84x-B, 86x-B, 87x-B, 88x-E 94x, 96x, 97x, 98x 182x 192x	2 oder 3	PWM Sign/Magnitude * PWM 50/50 16-Bit DAC
Schrittmotor	POSYS® 75x 85x 85x-B 95x 180x, 182x, 185x 190x, 192x, 195x	1	Takt & Richtung

* nur wenn 2 Phasen ausgewählt wurden

Sign/Magnitude PWM, 50/50 PWM und 16-Bit DAC Ausgangssignale verwenden einen Vorzeichen definierten numerischen Sollwert, allerdings in verschiedenen Formaten. Alle dieser Formate kodieren das Signal dahingehend, dass der numerische Wert einem Drehmoment- bzw. Geschwindigkeitssollwert entspricht, welche die POSYS® an den Verstärker ausgibt.

Takt & Richtung arbeitet auf eine fundamental andere Art als die beiden vorhergenannten Methoden, weil bei dieser Methode keine Möglichkeit besteht eine PID-Regelung zu verwenden. Stattdessen unterstützt Takt & Richtung Schrittmotortreiber die entweder einen Takt für die positive Richtung oder einen Takt für die negative Richtung erhalten. Der Verstärker selbst ist voreingestellt auf die Motorverstärkung für die es meistens verschiedene Einstellmöglichkeiten gibt.

Das Kommando SetOutputMode definiert welches der zur Verfügung stehenden Ausgabemethoden verwendet wird. Das Kommando GetOutputMode liefert den programmierten Wert zurück.

¤

Sign/Magnitude Impulsbreiten Modulierte Signalausgabe

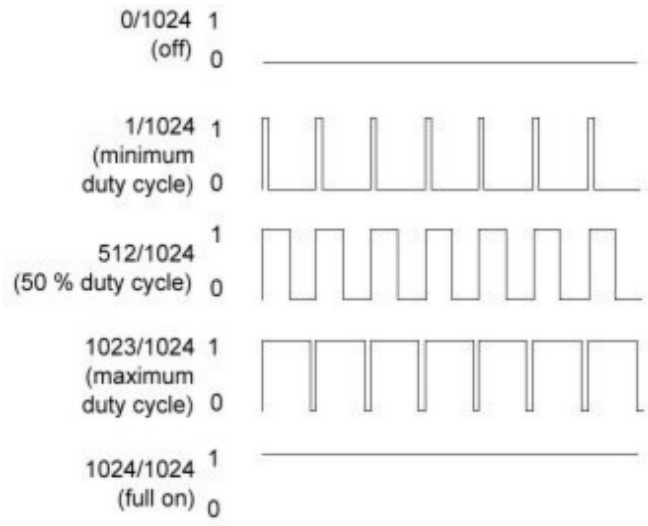
Sign/Magnitude PWM

In diesem Modus werden 2 Ausgänge verwendet um den Motor anzutreiben, eine für jede Motorphase. Ein Ausgang liefert das impuls-breiten-modulierte Signal welches von 0% bis 100% sein kann. Dieses Signal steht für die maximale Magnitude für den gewünschten Drehmomentsollwert. Ein Signal auf diesem Ausgang bedeutet, dass auf die Motorwicklung eine Spannung anliegen soll. Ein zweiter Ausgang definiert ob das Motorausgangsignal für eine positive oder für eine negative Drehrichtung sorgen soll.

In diesem Modus ist der Ausgang 1 Teil von 2048. Die Auflösung des Motorbefehlsregisters ist skaliert auf Werte von -32768 bis +32767 um den PWM-Ausgabebereich abzudecken.

Z.B. wenn das Ausgabesignal für eine Achse der POSYS® +12345 sein soll, dann wird das Richtungssignal high sein und das impuls-breiten-modulierte Signal wird mit folgendem Zyklus arbeiten: 2048*12345/32768 = 771,56 = 772. Dies bedeutet, dass das impuls-breiten-modulierte Signal für 772 Zyklen high sein wird und für die restlichen 1276 Zyklen low. Für die negative Richtung (-12345) würde das impuls-breiten-modulierte Signal das Gleiche bleiben aber das

Richtungssignal wäre stattdessen low.



Typisches Schema für impuls-breiten-modulierte Signale

Set/GetOutputMode

¤

50/50 Impulsbreiten Modulierte Signalausgabe

50/50 PWM

In diesem Modus wird nur ein Ausgang verwendet pro Motorausgang oder Motorphase. Dieser Ausgang trägt das von einem variablen Arbeitszyklus abhängigen impuls-breiten-modulierte Signal, dass dem vorhergehenden beschriebenen Signal sehr ähnelt. Im Gegensatz bedeutet aber ein Zyklus von 50% high und 50% low, und dass der Drehmomentsollwert wenn er den Wert 0 darstellt den Motor in der Position halten soll. Negative Fahrtrichtungen werden damit bewerkstelligt, dass der Zyklus weniger als 50% beträgt und positive Richtungen werden gefahren wenn der Zyklus einen Wert höher als 50% hat. In diesem Modus bedeutet 100% Ausgabewert, volle positive Richtungsbewegung und 0% volle negative Richtungsbewegung.

Z.B. wenn das Ausgabesignal welches der Controller ausgibt für eine bestimmte Phase +12345 ist, dann sieht die Ausgabeformel so aus: 1024+1024*12345/32768 = 1409,78 = 1410. Für 1410 Zyklen ist das impuls-breiten-modulierte Signal high und für die verbleibenden 638 Zyklen low. Wenn der gewünschte Wert aber -12345 ist 1024+24*-12345/32768 = 638,2 = 638, dann ist das impuls-breiten-modulierte Signal für 638 Zyklen high und low für die verbleibenden 1410 Zyklen.

50/50 PWM-Signale werden mit 2 verschiedenen Verstärkerversionen verwendet. Wenn ein bürstenloser Permanentmagnetmotor angetrieben werden soll ist das impuls-breiten-modulierte Signal mit Half Bridge Treiber angeschlossen.

Wenn ein DC bürstenbehafteter Motor verwendet wird, wird ein Half Bridge Verstärker benutzt, allerdings ist das Magnitudensignal von der Half Bridge immer eingeschaltet, und der Magnitudenausgang der POSYS® wird mit dem Vorzeicheneingang von der Half Bridge verbunden. Diese alternative Methode eine Half Bridge anzusteuern ist ab und zu nützlich in Situationen in denen man während der Abbremsrampe mit Rück-EMF rechnen muss, das die standardmäßigen Vorzeichenmagnitudensysteme benutzt.

Set/GetOutputMode

¤

16 Bit DAC Signalausgabe

16-bit DAC

In diesem Modus wird das Ausgabesignal direkt auf den Peripheriebus der POSYS® ausgegeben. Das digitale Signal wird mit Hilfe eines DAC in ein analoges Signal umgewandelt. Die POSYS® schreibt dieses Signal nacheinander auf jede aktivierte Achse im Rythmus der Servoloop Updaterate. Z.B. bei einer 4-Achsen POSYS® 19x4 Karte, bei der alle 4 Achsen aktiviert sind und die konfiguriert sind ein analoges Signal auszugeben, wäre die Standardzykluszeit 256 µs. Auf jede Achse wird innerhalb dieser Zeit einmal geschrieben und zwar in 64 µs-Schritten.

Für 1- bzw. 2-phasige Motoren wird ein DAC-Ausgang für jede Phase benutzt. Für 3-phasige Motoren werden nur 2 DAC-Ausgänge verwendet. Die dritte Phase wird immer ein analoges Signal sein gleich -1*(P1+P2) wobei P1 das Ausgabesignal für Phase 1 ist und P2 das Ausgabesignal für Phase 2. Im Bedarfsfall kann dieses dritte Phasensignal bei der Benutzung eines invertierenden und kummulierenden Verstärkers in der externen Schaltungsanordnung realisiert werden. Im Allgemeinen, obwohl dies nicht notwendig ist, da die Mehrzahl von 3-Phasen Standardverstärkern zwei Phasen akzeptiert und intern das Dritte konstruieren.

Die Auflösung des Analogsignals ist 16 Bit. Dieser Wert wird um 8000h versetzt ausgegeben, so wird ein Wert von 0 dem negativsten Wert entsprechen. Ein Wert von 8000h entspricht einem Ausgabewert von Null, und ein Wert von 0FFFFh entspricht dem positivsten Ausgabewert.

Set/GetOutputMode

¤

Hostkommunikation

Überblick

Überblick

Die POSYS® kommuniziert durch einer von bis zu drei möglichen Ports mit dem Host: entweder durch den bi-direktionalen Parallelport (PCI- oder ISA-Bus), einen asynchronen seriellen Port, oder via CAN 2.0B.

Die POSYS® akzeptiert Kommandos vom Host in einem Paketformat. Durch das Senden von Kommandos durch den Host kann das Verhalten der Positioniersteuerung entsprechend kontrolliert und der Status der POSYS® und der Motoren überwacht werden.

¤

Paralleler Kommunikationsport

Paralleler Kommunikationsport

Der bi-direktionale parallele Port ist dahingehend konfiguriert um im folgenden Modus zu operieren:

16-bit Modus	Die POSYS® übermittelt Kommandos und erhält Daten als volle 16 Bit Datenwörter, dabei d kompletten 16-bit Datenpfad des PC -Busses (PCI oder ISA) ausnutzend.

Dieser Modus ist Standard bei allen POSYS® Motion Controllern.

¤

Serieller Kommunikationsport

Serieller Kommunikationsport

Zusätzlich zum parallelen Port unterstützen die POSYS® (ausser 7xx/8xx Serie) eine asynchrone serielle Verbindung. Dieser serielle Port kann mit Baudraten zwischen 1200 und 416667 betrieben werden, dabei werden verschiedene Protokolle unterstützt (Punkt-zu-Punkt, Multi-Drop Mode usw., Näheres siehe unten).

Folgende Verbindungen werden auf den Karten verwendet.
POSYS® 8xx-B: 2-reihiger, 6-poliger Stecker (2 mm)
Der Gegenstecker ist eine 2-reihige, 6-polige Buchse (2 mm)
POSYS® 18xx/9xx/19xx: Molex 5-poliger PicoBlade SMT style connector part no.: 53261-0590 oder 53261-0571 (blei-frei)
Der Gegenstecker ist der Molex 5-polige Stecker: Molex part no. 51021-0500

Konfiguration

Nach einem Reset liest die POSYS® einen 16 Bit Wert vom Peripheriebus (Adresse: 200h), welches verwendet wird um die Standardkonfiguration des seriellen Ports zu setzen. Falls der serielle Port für die Kommunikation verwendet werden soll, dann muss externe Zusatzhardware verwendet werden um den Zugang zu dekodieren und eine passende Konfiguration wie unten beschrieben, zu verwenden. Dies geschieht über ein optional von uns erhältliches Zusatzkabel.

Alternativ, falls es nicht wünschenswert ist externe Hardware hinzu zufügen, kann der externe Datenbus der POSYS® auf „high“ gesetzt werden indem „high-value registers“ (z.B. 10 kOhm) verwendet werden. Dies veranlasst die POSYS® den Wert 0FFFFh an der Adresse 200h zu lesen. Wenn dieser Wert von der POSYS® gelesen wird, wird sie veranlasst mit folgenden Defaultwerten an der seriellen Schnittstelle zu starten: 57600 Baud, 0 Parity, 2 Stopp Bits, Point-to-Point Mode.

Der Konfigurationswort ist folgendermaßen aufgeschlüsselt um andere Konfigurationen zu laden:

Bit	Parameter	Indikationen	
0-3	Transmission rate selector	0 1 2 3 4 5 6 7	1200 bps 2400 bps 9600 bps 19200 bps 57600 bps* 115200 bps* 250000 bps 416667 bps
4-5	Parity selector	0 1 2 3	None Odd parity Even parity Reserved (nicht verwenden)
6	Number of stop bits	0 1	1 stop bit 2 stop bits
7-8	Ptotocol type	0 1 2 3	Point-to-point Reserved (nicht verwenden) Multi-drop (address bit mode) Multi-drop (idle line mode)
9-10	Reserves		
11-15	Multi-drop address selector Should be zero in point-to-point mode	0 1 ... 31	Address 0 Address 1 Address 31

*Um bei dieser Geschwindigket eine sichere Synchronisation zu gewährleisten, sollten 2 Stoppbits eingestellt werden.

Kontrollsignale

Drei Signale SrlXmt, SrlRcv und SrlEnable sind für die Kommunikation über den seriellen Port notwendig. SrlXmt kodiert Daten vom Motionprozessor zum Host. SrlRcv empfängt Daten vom Host an den Motionprozessor. SrlEnable wird active (high) wenn Daten im Multi-drop Mode verschickt werden. Dieses Signal kann mit dem OutputEnablePin des seriellen Buffer ICs verbunden werden um Tri-Stating der Datenleitung zu erlauben wenn die Übermittlungsleitung des seriellen Busses nicht verwendet wird. In Point-to-Point Mode ist SrlEnable immer active (high). Die Basiseinheit für die serielle Datenübertragung (Empfang und Versand) ist das Byte. Jedes Byte besteht aus folgenden Einheiten:

- 1 Startbit
- 8 Datenbits
- 1 optionaler even/odd Paritybit
- 1 oder 2 Stoppbits
- 1 extra Bit welches Adress und Datenbits auseinanderhält (Address-Bit Stil nur im Multi-Drop Mode)

Kommandoformat

Das Kommandoformat, dass verwendet wird um eine Kommunikation zwischen Host und POSYS® zu ermöglichen, besteht aus einem Kommandopaket, dass vom Host gesendet und von der POSYS® mit einem Antwortpaket beantwortet wird.

Kommandoakete vom Host enthalten folgende Felder:

Feld	Byte	Beschreibung
Address	1	Ein Byte um zu identifizieren an welche POSYS das Kommandopaket geschi wird. Diese Feld sollte im Point-to-Point Mode immer Null sein.
Checksum	2	Ein 1 Byte Wert um Paketdaten zu verifizieren. Für Beschreibung siehe unte
Instruction code	3-4	Eine 2 Byte Instruktion, das obere Byte wird zuerst gesendet (Achsennummer). Die Kommandostruktur ist dieselbe die im Parallelkommunikationsmodus verwendet wird.
Data	5-	Null bis 6 Bytes an Daten, das meist signifikante Byte zuerst. Die individuel Kommandobeschreibungen sind im Detail im Programmierhandbuch beschrieben.

Als Antwort sendet die POSYS® folgendes Datenpaket:

Feld	Byte	Beschreibung
Status	1	Null falls das Kommando erfolgreich ausgeführt wurde, ansonsten wird ein Fehlercode der den Fehler näher spezifiziert gesendet (siehe HostIO Errors)
Checksum	2	Ein 1 Byte Checksum Wert um die Paketintegrität zu verifizieren.
Data	3	Null bis 6 Bytes an Daten. Falls ein Fehler auftrat werden keine Bytes gesendet (der Status-Byte is nicht Null). Falls kein Fehler auftrat, dann wird die Anzahl Bytes davon abhängen auf welches Kommando die POSYS® antworten sollte. Das meist signifikante Byte wird immer zuerst gesendet.

Host I/O Errors

Der Motionprozessor vollführt eine Anzahl Überprüfungen abhängig vom Kommando, dass an die POSYS® gesendet wurde. Diese Überprüfungen erhöhen die Sicherheit des Positioniersystems indem offensichtlich inkorrekte Kommandodatenwerte eliminiert werden. All jene Überprüfungen die mit Host I/O Kommandos assoziiert sind, werden als Host I/O Errors bezeichnet. Der Status Byte im Antwortpaket kann eines der folgenden Fehlercodes enthalten:

Code	Indikation	Grund
0	No error	Kein Fehler
1	Pilot reset	Standardwert nach einem Reset oder Power-On.
2	Invalid instruction	Das Kommando ist im vorliegenden Kontext ungültig, oder ein illegaler Instruktionscode wurde entdeckt.
3	Invalid axis	Die Achsennummer in den oberen Bits des Kommandowortes wird vom C h nicht unterstützt.
4	Invalid parameter	Der Parameterwert der zur POSYS® gesendet wurde is in einem nicht akzeptablen Bereich.
5	Trace running	Es wurde ein Kommando gesendet, dass den Status des Tracemechanismus ändern würde, während der Trace am laufen ist. Kommandos die diesen Fehler zurückliefern können, sind Aufrufe von SetTraceVariable, SetTraceM & SetTracePeriod.
6	Reserved	
7	Block bound exceeded	1. Der Wert für SetBufferLength oder SetBufferStart würde einen Speicherblock generieren der größer ist als erlaubt (400h – 7FFFFFFFh) 2. Entweder SetBufferReadIndex oder SetBufferWriteIndex haben einen IndexWert größer oder gleich der Blocklänge gesendet.
8	Trace zero	SetTraceStart Immediate wurde ausgeführt, aber die momentane Länge de TraceBuffers ist gleich 0.
9	Bad checksum	Die Checksum die verglichen und von der POSYS® zurückgeschickt wurde, unterschiedlich der der vom Host gesendet wurde.
Bh	Negative velocity	Es wurde versucht eine Achse mit einer negativen Geschwindigkeit zu verfahren, obwohl die Achse nicht im Geschwindigkeitsmodus ist.
Ch	S-curve change	Die Achse vollführt gerade ein S-Kurvenprofil und es wurde der Versuch gemacht Profilparameter zu ändern. Dies ist nicht erlaubt.
Dh	Limit event pending	Ein Endschalter wurde angefahren.
Eh	Move into limit	Es wurde der Versuch gemacht eine Bewegung auszuführen ohne zuerst die Endschalterbits im Event Status Register zurück zu setzen.

Checksummen

Beides, sowohl das Kommando als auch die Antwort enthalten ein Checksummen Byte. Die Checksumme wird verwendet um Übertragungsfehler zu ermitteln und erlaubt die POSYS® fehlerhafte Datenpakete zu identifizieren und abzulehnen.

Anders als beim Parallelport sind Checksummen bei seriellen Schnittstellen obligatorisch. Alle Kommandopakete die zur POSYS® geschickt wurden und die ungültige Checksummenwerte beinhalten, werden nicht weiter gegeben und ein Datenpaket mit einem Error Status Code wird zurückgeschickt.

Die serielle Checksumme wird kalkuliert indem alle Bytes addiert werden (mit Ausnahme des Checksummen Bytes), anschliessend wird das 2-er Kompliment des Ergebnisses ermittelt. Die acht niederwertigen Bits dieses Ergebnisses werden als Checksumme verwendet. Um eine korrekte Checksumme zu prüfen, sollten alle Bytes eines Pakets summiert werden (inkl. des Checksummen Bytes) und falls das Ergebnis der niederwertigen acht Bits = 0 ist, dann ist die Checksumme gültig.

Übertragungsprotokolle

Die POSYS® unterstützt die Möglichkeit zusätzliche Karten am seriellen Bus anzuhängen, ähnlich einer Kette, oder einem Netzwerk von POSYS®s die mit den gleichen seriellen Hardwaresignalen kommunizieren.

Drei Möglichkeiten bestehen mit der seriellen Schnittstelle Zeitprobleme, Übertragungsfehler und andere Vorgänge die während der seriellen Übertragung auftreten können, zu lösen. Da gibt es die Punkt-zu-Punkt Übertragung, falls nur eine Einheit verwendet wird, Multi-Drop Address-Bit Mode und Multi-Drop Idle-Line Mode, falls mehrere Karten an der seriellen Schnittstelle hängen. Die nächsten Sektionen beschreiben diese 3 Übertragungsprotokolle.

Punkt-zu-Punkt Modus

Dieser Modus ist für die direkte serielle Kommunikation zwischen POSYS® und Host vorgesehen, wenn eine direkte serielle Verbindung besteht zu einer Karte. In diesem Modus wird der Addressbyte nicht von der POSYS® verwendet (ausser zur Kalkulation der Checksumme) und die POSYS® antwortet auf alle Kommandos die vom Host kommen.

Wenn in diesem Modus, gibt es keine zeitkritischen Anforderungen an die Daten die übermittelt werden innerhalb eines Paketes. Die Menge an Daten die ein Kommandopaket beinhaltet, werden vom Kommando bestimmt im Datenpaket. Mit jedem Kommadopaket ist eine spezifische Menge an Daten verbunden/assoziiert. Wenn die POSYS® ein Kommandokode erhält wartet sie bis alle Daten empfangen wurden, ehe das Kommando ausgeführt wird. Auch die zurückgelieferte Menge an Daten die von einem Kommando angefordert wurde, ist bestimmt vom Kommandokode. Nach Ausführung eines Kommandos, wird die POSYS® ein Datenpaket in der erforderlichen Größe und Länge zurückliefern.

Wenn im Punkt-zu-Punkt Modus, gibt es für die POSYS® keine direkte Möglichkeit den Anfang eines neuen Datenpaketes zu unterscheiden, außer im Zusammenhang. Daher ist es für den Host wichtig in einem synchronisierten Zustand mit der POSYS® zu verbleiben wenn Daten gesendet und empfangen werden. Um sicher zu stellen die POSYS® verbleibt synchronisiert, wird empfohlen im Host eine Zeitspanne zu definieren auf die sie warten soll wenn Datenpakete von der POSYS® erwartet werden. Die empfohlene minimale Zeitspanne ist die Zeit die benötigt wird ein Byte bei der gewählten Baudrate plus 1 Millisekunde zu senden. z.B., bei einer Baudrate von 9600 Baud/sek. benötigt jedes Bit 1/9600 Sek. für die Übertragung. Ein typischer Byte besteht aus 8 Datenbits plus 1 Startbit und einem Stoppbit. Daher benötigt ein Byte gerade etwas mehr als 1 Millisekunde und die empfohlene Zeitspanne ist 2 Millisekunden. Falls es zu einer zeitlichen Überschneidung von zu empfangenden Datenbytes kommt während der Host auf Datenpakete wartet, dann sollte der Host davon ausgehen sich außerhalb der Synchronisation zu befinden. Um die Synchronisation wieder herzustellen, sollte der Host ein Byte ohne Daten schicken und auf die Antwort warten. Dieser Prozess sollte solange wiederholt werden, bis ein Datenpaket von der POSYS® empfangen wurde. An diesem Punkt werden Host und POSYS® wieder synchronisiert.

Multi-drop Protokolle

Von der POSYS® werden 2 verschiedene Multi-drop Protokolle unterstützt. Diese Modi werden vornehmlich dann auf einer seriellen Schnittstelle verwendet wenn ein einzelner Host mit mehreren POSYS® (oder anderen untergeordneten Geräten) kommunizieren soll. In diesem Modus wird zuerst ein Addressbyte benutzt, welches definiert für welches Gerät oder welche Einheit (POSYS®) das Datenpaket gedacht ist. Nur die adressierte Einheit wird auf das Gerät antworten bzw. reagieren. Daher ist es wichtig die Adressen der POSYS® sorgfältig auszuwählen (serielle Konfigurationswort, wie oben beschrieben) und das Addressbyte für jedes Kommandopaket einzubinden, dass für die POSYS® gedacht ist.

Da die Adresse, die einem Kommandopaket vorangeht, gedacht ist, die Antwort einer POSYS® im Multi-drop Mode zu aktivieren bzw. zu deaktivieren, muss das Multi-drop Protokoll eine Methode beinhalten die Synchronisation zw. dem Host und den verschiedenen POSYS® aufrecht zu erhalten, da es in einer solchen Umgebung schwierig ist erneut die Synchronisation herzustellen. Die beiden Multi-drop Protokolle differieren in der Methode wie sie die Synchronisation kontrollieren.

Bitte beachten Sie, dass die Multi-drop Protokolle auch in einer Punkt-zu-Punkt Konfiguration verwendet werden können, solange der Host immer das Addressbyte für jedes Kommandopaket mit vorne wegschickt und auch allen anderen Gesetzmäßigkeiten für das selektierte Protokoll folgt. Diese Methode der Handhabung erlaubt dem Host mit Sicherheit in einem synchronisierten Zustand zu verbleiben, ohne die Notwendigkeit irgendwelche Zeitspannen und Resynchronisationsmethoden zu implementieren.

Idle-line Protokoll

Bei der Verwendung dieses Protokolls werden von der POSYS® strenge deterministische Auflagen an die Daten, die Teil eines Kommandos sind, gestellt. In diesem Modus interpretiert die POSYS® das 1. empfangene Byte nach einer Periode der Untätigkeit als den Beginn eines neuen Pakets. Alle bis dahin empfangenen Daten werden verworfen.

Die Timeout Periode ist gleich der Zeit die erforderlich ist 10 Bits an seriellen Daten mit der konfigurierten Baudrate zu senden (grob 1 Millisekunde bei 9600 Baud). Falls eine Verzögerung zwischen empfangenen Bytes eines Datenpakets in dieser Länge auftritt, werden die bereits empfangenen Bytes verworfen und das 1. Zeichen nach der Verzögerung als Addressbyte eines neuen Paketes gewertet.

Address Bit Protokoll

Bei diesem Protokoll wird jedes übertragene Byte, dass von der POSYS® empfangen oder versendet wird, ein zusätzliches

Bit enthalten (nach dem letzten Datenbit). Dieses Bit wird für die Identifikation des Addressbytes eines Paketes verwendet. Jedes Byte, dass von der POSYS® empfangen wird und bei dem dieses Bit gesetzt ist, wird als den Beginn eines neuen Kommandopakets interpretiert. Falls die POSYS® dabei war Daten eines früheren Kommandopakets zu empfangen, werden jene Daten verworfen.

¤

Controller Area Network CAN2.0B

Überblick

Die POSYS® 18xx/19xx können in ein CAN 2.0B Netzwerk integriert werden und mit anderen CAN Knoten koexisitieren aber nicht kommunizieren. Die POSYS® verwendet CAN um Kommandos zu empfangen, Antworten und optional auch asynchron Ereignisbenachrichtigungen zu senden. Jede Meldung hat folgende Adresse:

Meldung	CAN Adresse
Kommando empfangen	0x600 + nodeID
Kommando antworten	0x580 + nodeID
Ereignisbenachrichtigung	0x180 + nodeID

CAN Knoten kommunizieren mit Hilfe von Botschaften. Jede Botschaft kann bis zu 8 Bytes an Daten beinhalten. Der CAN Protokoll ist in der Lage automatisch Übertragungsfehler zu korrigieren, daher ist die Verwendung einer Checksumme nicht notwendig wie bei der seriellen Datenübertragung.

Die Datenformate der Meldungstypen werden in Termini von Bytesequenzen für die parallele Schnittstelle ausgedrückt. Kommandos haben unterschiedliche Längen; das POSYS® Programmierhandbuch zeigt dies auf.

Bytes, die immer vorhanden sind wenn ein Kommando ausgeführt wird, sind in der Spalte „Notwendig?“ entsprechend markiert. Die korrespondierenden Bytesequenzen für die 3 Meldungstypen sind wie folgt:

Kommando empfangen Message data byte	Notwendig?	Korrespondierendes Parallelbyte
1	J	Command word, high byte
2	J	Command word, low byte
3	N	1st data word, high byte
4	N	1st data word, low byte
5	N	2nd data word, high byte
6	N	2nd data word, low byte
7	N	3rd data word, high byte
8	N	3rd data word, low byte

Kommando empfangen Message data byte	Notwendig?	Korrespondierendes Parallelbyte
1	J	Command word, high byte
2	J	Command word, low byte
3	N	1st data word, high byte
4	N	1st data word, low byte
5	N	2nd data word, high byte
6	N	2nd data word, low byte
7	N	3rd data word, high byte
8	N	3rd data word, low byte

Event Benachrichtigung Message data byte	Notwendig?	Dateninterpretation
1	J	Axis number (1-4)
2	J	event status register value, high byte
3	J	event status register value, low byte

¤

Konfiguration der CAN2.0B Schnittstelle

Konfiguration der CAN-Schnittstelle

Die CAN-Schnittstelle der POSYS® wird mit dem Kommando SetCANMode konfiguriert. Dieses Kommando wird benötigt um die CAN NodeID einer spezifizierten POSYS® (0-127) zu setzen, als auch die Übertragungsrate des angeschlossenen CAN-Netzwerkes. Im Folgenden werden die unterstützten Übertragungsraten aufgelistet:

SetCanMode Encoding	CAN Transmission Rate (bps)
0	1000000
1	800000
2	500000
3	250000
4	125000
5	50000
6	20000
7	10000

Die Standardwerte beim Einschalten sind für die verschiedenen Versionen die selben. Andere Standardeinschaltwerte sind möglich und können kundenspezifisch vor Auslieferung voreingestellt auf den Karten fest einprogrammiert werden. Diese kundenspezifische Voreinstellung muss vom Kunden vor der Bestellung angegeben werden.

Die Standardwerte sind wie folgt:

Parameter	Standardwert
Übertragungsrate	500000
CAN nodeID	0

Die Standardwerte können durch entsprechende Softwarekommandos jederzeit überschrieben werden. Ein SoftReset wird diese Werte nicht zurücksetzen, sie bleiben erhalten und werden vom Motionprozessor wieder eingelesen und als neue Standardwerte verwendet. Nach einem HardReset oder einer Grundinitialisierung nach dem Einschalten werden wieder die fest voreingestellten Standardwerte wieder aktiv.

¤

CAN Ereignis Benachrichtigung

CAN Ereignis Benachrichtigung

Wenn die Kommunikation über die CAN-Schnittstelle erfolgt, kann die POSYS® optional Nachrichten senden, wenn im Event Status Register entsprechende Bits als aktiv gesetzt sind. Diese Möglichkeit steht in direktem Zusammenhang mit der Interrupt-Funktion des Motion Control Prozessors, wenn die parallele Schnittstelle (Board-intern) verwendet wird. Diese Meldungen werden mit der CAN-Adresse 0x180 + nodeID versendet.

Diese Funktionalität wird mit dem Kommando SetInterruptMask spezifiziert und kontrolliert. Für jedes Bit in der Benachrichtigungsmaske, dass als ON markiert ist, wird eine CAN-Nachricht generiert sobald das korrespondierende Bit im Event Status Register „1“ wird. Im Folgenden sind die verfügbaren Maskenbits gelistet:

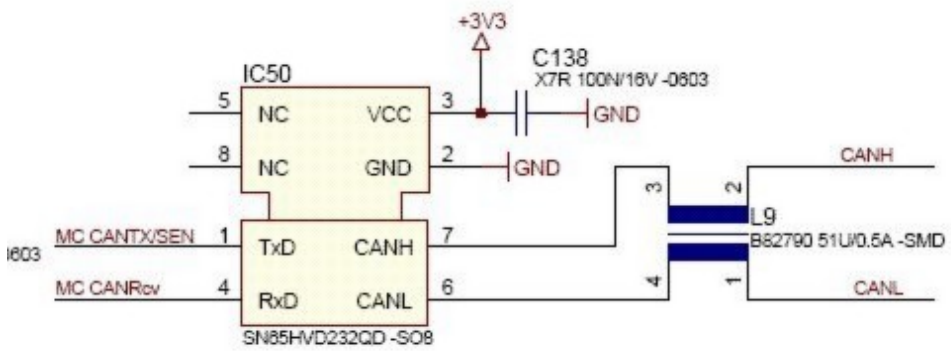
POSYS® Ereignis	Maskenbit
Motion complete	0001h
Wrap-around	0002h
Breakpoint 1	0004h
Capture received	0008h
Motion error	0010h
In positive limit	0020h

POSYS® Ereignis	Maskenbit
In negative limit	0040h
Instruction error	0080h
Commutation error	0800h
Breakpoint 2	4000h

¤

Elektrisches Diagramm für CAN Schnittstelle

Elektrische Diagramme für CAN-Schnittstelle



CAN Interface für POSYS® 1800/1900 Serie Motion Controller

Die POSYS® 1800/1900 verwendet den TI SN65HVD232QD-SO8 CAN Transceiver. Für weitere Details empfehlen wir das Datenblatt, dass sie bei Texas Instruments herunterladen können. Die CANH und CANL Verbindungen werden mit Hilfe der mit CON3 bezeichneten Molex Stecker auf der POSYS 1800/1900 Karte bewerkstelligt.

Die Steckverbindung auf der Karte ist ein Molex Stecker (Artikelnummer: 53398 – 0290). Das dazu gehörige Gegenstück ist eine Molex Buchse (Artikelnummer: Molex 51021-0200, Crimpkontakt: Molex 50079-8100).

¤

Verwendung des On-Board Speichers

Speicherkonfiguration

Speischerkonfiguration

POSYS® 18xx/19xx

Die POSYS® ist in der Lage On-Board Speicher zu adressieren um Tracedaten zu speichern. Zusätzlich kann es verwendet werden um andere Daten zu hinterlegen, wie z.B. Profilsequenzen (ExternalProfileMode).

Jede Größe des RAM kann von 0KB bis zur maximalen Auslastung adressiert werden. Die Gesamtmenge an Speicher, die Verwendung finden soll, muss entsprechend kalkuliert werden.

Die Adress- und Datenbusse der POSYS® werden verwendet, um mit dem On-Board Speicher verbunden zu werden. Der Controller verwendet 15 Adressleitungen für On-Board Speicher welches dem Controller erlaubt direkt auf 32K Worte des Speichers zuzugreifen. Um einen größeren, adressierbaren Bereich zu ermöglichen, verwendet der Controller einen Memory Page Selector (gemappt in den peripheren Adressbereich bei 2000h), welches Zugriff auf 64K Pages erlaubt. Dies erlaubt den maximal adressierbaren Speicher von bis zu 2048MB Worte mit je 16-Bit Größe.

Diese Organization des Speichers gibt dem Entwickler diverse Optionen in wie weit diese Funktionalität unterstützt werden soll:

Speichergröße	Bemerkungen
Kein	Der On-Board Speicherbereich ist optional. Diagnostische und andere Möglichkeite die den Speicher verwenden stehen nicht zur Verfügung.
0K bis 32K	Sämtlicher Speicher ist in der ersten Seite, daher ist es nicht notwendig Schreibvorgänge in einen externen Seitenzeiger (external page pointer) zu decodieren.
Mehr als 32K	Ein externer Seitenzeiger (external page pointer) muss verwendet werden.

POSYS® 7xx/8xx/8xx-B/9xx

Die POSYS® ist in der Lage On-board Speicher zu adressieren um Tracedaten zu speichern. Zusätzlich kann es verwendet werden um andere Daten zu hinterlegen, wie z.B. Profilsequenzen (ExternalProfileMode).

Jede Größe des RAM kann von 1K bis zur maximalen Auslastung adressiert werden, allerdings werden die ersten 1024 Worte des Speichers für den Controller reserviert. Die Gesamtmenge an Speicher die Verwendung finden soll, muss entsprechend kalkuliert werden.

Die Adress- und Datenbusse der POSYS® werden verwendet um mit dem On-board Speicher verbunden zu werden. Der Controller verwendet 15 Adressleitungen für Speicher welches dem Controller erlaubt direkt auf 32K Worte des Speichers zuzugreifen. Um einen größeren, adressierbaren Bereich zu ermöglichen, verwendet der Controller einen Memory Page Selector (gemappt in den peripheren Adressbereich bei 2000h), welches Zugriff auf 64K Pages erlaubt. Dies erlaubt den maximal adressierbaren Speicher von bis zu 2048MB Worte mit je 16-Bit Größe.

Diese Organization des Speichers gibt dem Entwickler diverse Optionen in wie weit diese Funktionalität unterstützt werden soll:

Speichergröße	Bemerkungen
Kein	Der On-board Speicherbereich ist optional. Diagnostische und andere Möglichkeite die den Speicher verwenden, stehen nicht zur Verfügung.
1K bis 32K	Die ersten 1K (1024 Worte) werden vom C ontroller reserviert. Sämtlicher Speicher ist in der ersten Seite, daher ist es nicht notwendig Schreibvorgänge in einen externen Seitenzeiger (external page pointer) zu decodieren.
Mehr als 32K	Ein externer Seitenzeiger (external page pointer) muss verwendet werden. Das er Kilobyte Worte der Seite 0 ist reserviert für den C ontroller.

¤

Memory Page Pointer

Memory Page Pointer

Falls ein Adressbereich von mehr als 32K Worte gewünscht wird, dann muss ein externer Seitenregister erstellt/unterstützt werden.

Ehe der Controller in den On-Board Speicherbereich schreibt, schreibt er ein 16-Bit Seitennummer in den peripheralen Adressbereich 2000h. Dieser Wert sollte festgehalten werden und als die oberen 16-Bit der tatsächlichen 31-Bit Speicheradresse verwendet werden.

Der Regler wird niemals versuchen von diesem Seitenzahlenregister zu lesen, also ist auch keine Notwendigkeit Lesevorgänge zu unterstützen.

Obwohl der Wert der vom Controller geschrieben wird 16-Bit ist, gibt es keine Notwendigkeit irgendwelche Bits zu latchen die nicht benötigt werden. Z.B. falls 128K Worte an Speicher zur physischen Adresse 0-1FFFFh gemappt wurden, dann werden 4 Seiten Speicher benötigt (4 x 32K = 128K). Dies entspricht den zwei niedrigen Bits des Seitenregisters. Wenn der Controller einen 16-Bit Wert in den peripheren Adressbereich 2000h schreibt, sollten die zwei niedrigsten Bits gelatcht und als Adressenbits 15 und 16 verwendet werden, wenn auf On-Board Speicher zugegriffen wird.

¤

On-Board Speicher Signal Dekodierung

On-Board Memory Signal Decoding

Die POSYS® Controller unterstützen einen On-Board RAM Speicherbus um Schreib- und Lesevorgänge zu erlauben. Der Bus besteht aus 4 Kontrollsignalen; ~RAMSlct, W/~R, R/~W, und ~WriteEnbl. Zusätzlich gibt es 15 Adressleitungen (Addr0-Addr14) und 16 Datenleitungen (Data0-Data15).

¤

On-Board Memory Read

On-Board Memory Read

Um vom On-Board Speicher lesen zu können, lädt zuerst der Controller den Zeiger auf den Seitenspeicher. Um dann lesen zu können wird der Controller das ~RAMSlct, ~Strobe und W/-R Signal auf low level setzen. Ausserdem wird der Controller den Adressbus auf high oder low setzen, je nach dem auf welchen bestimmten Speicherbereich zugegriffen wird.

ReadBuffer, WriteBuffer

¤

On-Board Memory Write

On-Board Memory Write

Um in den On-Board Speicher schreiben zu können lädt zuerst der Controller den Zeiger auf den Seitenspeicher. Um dann schreiben zu können wird der Controller das ~RAMSlct, ~Strobe, W/-R und ~WriteEnbl Signal auf low level setzen. Ausserdem wird der Controller den Adressbus auf high oder low setzen, je nach dem auf welchen bestimmten Speicherbereich zugegriffen wird und es wird auch das Datenwort auf den Datenbus übergeben, der vom Regler geschrieben werden soll.

ReadBuffer, WriteBuffer

¤

On-Board Memory Buffer

On-Board Memory Buffer

Die POSYS® stellt einige Kommandos zur Verfügung die Schreib- und Lesevorgang für den On-Board Speicher ermöglichen. Mit diesen Kommandos können bis zu 16 Speicherbuffer definiert werden. Ein Buffer beschreibt einen lückenlosen Speicherblock indem es die Basisadresse und die Blocklänge definiert.
Nachdem ein Block definiert wurde, können Daten in diesen Block geschrieben und auch wieder rausgelesen werden.

Beim Definieren von Speicherbuffern wird der On-Board Speicherbereich als eine Sequenz von 32-Bit Speicherpositionen behandelt. Jeder 32-Bit Wert kann bis zu 2x 16-Bit Speicherpositionen des physikalischen Speichers inne haben.
Die Bufferadressen und -längen bestehen beide aus 32-Bit Werten und müssen daher verdoppelt werden um genügend physikalischen Speicher zur Verfügung zu haben.

Der Controller wird alle Werte für die Basisadresse und -länge zulassen solange es legale Adressen sind. Legale Adressbereiche rangieren von 200h (nur gültig für POSYS® 700/800/800-B/900; welches mit der physikalischen Adresse 400h zusammenfällt gerade nach dem 1K reservierten Block) bis zu 3FFFFFFFh (welches der physikalischen Adresse 7FFFFFFEh entspricht). Die POSYS® 1800/1900 haben keine Speicherbeschränkungen. Der gesamte Adressraum ab 0h darf verwendet werden.

Es sollte unbedingt vermieden werden nicht existenten Speicher zu adressieren da dies unvorhersehbare Folgen im Verhalten des Controllers zur Folgen haben kann.

Zusätzlich zur Basisadresse und -länge bewahrt jeder Speicherblock einen Schreib- und Leseindex.
Dem Leseindex kann ein Wert zugewiesen werden zwischen 0 und L-1 (L ist die Bufferlänge). Es definiert die Position von der der nächste Wert gelesen wird. Ganz ähnlich der Schreibindex der von 0 bis L-1 (L ist die Bufferlänge) rangiert und die Position definiert bei dem der nächste Wert geschrieben wird. Wenn ein Wert vom Speicherbuffer gelesen wird, wird automatisch der Leseindex inkrementiert, dabei den nächsten Wert zum lesen selektierend. Der Schreibindex is inkremental wann immer in den Buffer geschrieben wird.
Wenn jeder Index das Ende des Zwischenspeichers erreicht, wird es automatisch auf 0 zurückgestellt bei der nächsten Schreib-/Leseoperation.

Set/GetBufferFunction, Set/GetBufferLength

¤

On-Board Speicherkommandos

On-Board Speicherkommandos

Dieser Abschnitt beschreibt die Kommandos um auf den Speicherbereich zuzugreifen und Schreib- und Lesevorgänge zu vollziehen.

Kommando	Beschreibung
SetBufferStartbufferID, address	Setzt die Basisadresse des Zwischenspeichers. bufferID ist ein 16-Bit Integerwert im Bereich von 0-15, welches an gibt, welchen Zwischenspeicher es modifizieren soll, address ist ein 32-Bit Integerwert im Bereich von 200h bis 3FFFFFFFh, welches die neue Basisadresse des Zwischenspeichers definiert. Der C ontroller addiert address zur gegenwärtigen Zwischenspeicherlänge (wie in SetBufferLength definiert) um sicher zu stellen, dass der Zwischenspeicher den maximal adressierbaren Speicherbereich nicht überschreitet. Falls die Grenze überschritten wird, wird das Kommando ignoriert und ein Instruction Err Bit gesetzt.
GetBufferStartbuffered	Liefert die Basisadresse des abgefragten Zwischenspeichers
SetBufferLength bufferID, length	Definiert die Länge des Zwischenspeichers. bufferID ist ein Integerwert i Bereich von 0-15. length ist ein 32-Bit Integer im Bereich von 1 bis 3FFFFFFFh. Der Controller addiert length zur gegenwärtigen Zweischenspeicherbasisadresse (wie in SetBufferStart gesetzt) um siche zu stellen, dass der maximal zur Verfügung stehende Speicherbereich ni

Kommando	Beschreibung
	überschritten wird. Falls doch, wird das Kommando ignoriert und das Instruction Error Bit wird gesetzt.
SetBufferStart und SetBufferLength setzen den Bufferindex auf null zurück.	
GetBufferLength buffered	Liefert die Länge des spezifizierten Buffers.
SetBufferReadIndex bufferID, index	Setzt den Schreibindex für den spezifizierten Buffer. Index ist ein 32-Bit Integerwert von 0 bis length-1, wobei length die momentane Bufferlänge definiert. Falls index nicht innerhalb dieses Bereiches liegt wird es nicht gesetzt und ein Instruction Error Bit wird gesetzt.
GetBufferReadIndex bufferID	Liefert den Wert des Schreibindexes für den angegebenen Zwischenspeicher.
SetBufferWriteIndex bufferID, index	Setzt den Leseindex für den spezifizierten Buffer. index ist ein 32-Bit Integerwert von 0 bis length-1, wobei length die momentane Bufferlänge definiert. Falls index nicht innerhalb dieses Bereiches liegt wird es nicht gesetzt und ein Instruction Error Bit wird gesetzt.
GetBufferWriteIndex bufferID	Liefert den Wert des Schreibindexes für den angegebenen Zwischenspeicher.
ReadBuffer bufferID	Liefert einen 32-Bit Wert vom spezifizierten Zwischenspeicher zurück. Di Position von dem dieser Wert gelesen wird, wird bestimmt indem zum Leseindex die Basisadresse hinzuaddiert wird. Nachdem der Wert gelese wurde, wird der Leseindex inkrementiert. Falls das Ergebnis gleich der gegenwärtigen Bufferlänge entspricht wird der Leseindex auf Null zurückgesetzt.
WriteBuffer bufferID, value	Schreibt einen 32-Bit Wert in den spezifizierten Zwischenspeicher. Die Position wird bestimmt indem zum Schreibindex die Basisadresse hinzuaddiert wird. Nach dem der Wert geschrieben wurde, wird der Schreibindex inkrementiert. Falls das Ergebnis gleich der gegenwärtigen Bufferlänge entspricht, wird der Schreibindex auf Null zurückgesetzt.

¤

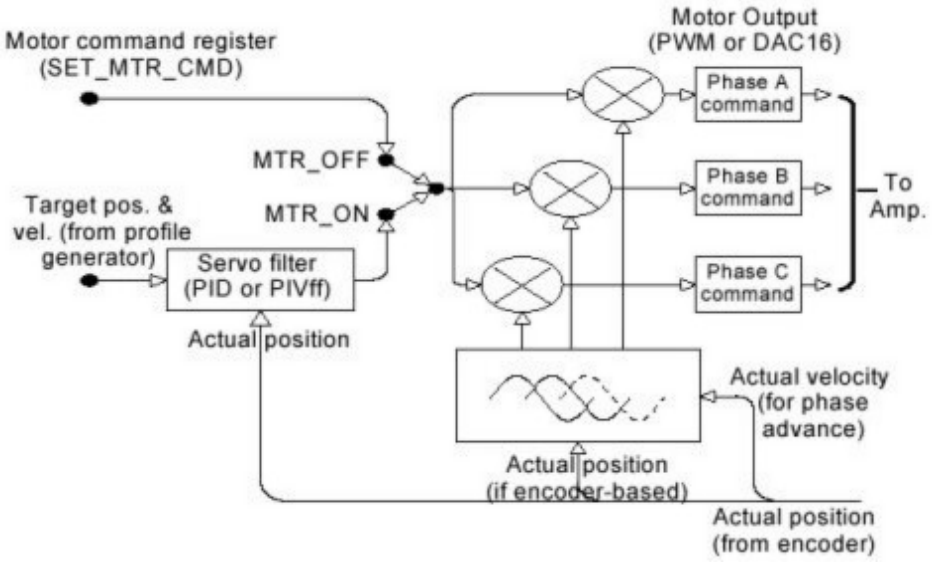
Sinus-Kommutierung (POSYS® Achsen in bürstenlosem Modus)

Überblick

Überblick

Die POSYS® unterstützen zusätzlich zur Bahnprofilerstellung und geschlossenem Servo-Regelkreis auch die Möglichkeit das Kommutierungssignal für 2- und 3-phasige bürstenlose Servomotoren zu generieren (Modell-abhängig). Dies erlaubt die gemischte Benutzung von bürstenlosen und bürstenbehafteten Servomotoren mit ein und dem selben Controller.

Das folgende Diagramm bietet einen Überblick über die Flusskontrolle der Sinuskommutierung einer Achse der POSYS® die im bürstenlosen Modus betrieben wird:



Der Kommutierungsteil des Controllers verwendet einen Eingang des Motorausgangssignals, entweder vom Servofilter oder vom Motorkommandoregister (abhängig ob die Achse im Closed Loop oder Open Loop Mode betrieben wird). Dieses vorkommutierte Kommandosignal wird mit Kommutierungswerten, hergeleitet von einer internen Sin/Cos Tabelle, multipliziert.

Der Kommutierungswinkel in der Sin/Cos Tabelle wird sowohl von der Enkoderposition als auch von Parametern die vom Hostprozessor stammen, bestimmt, die den spezifischen Enkoder ins Verhältnis zu den magnetischen Polen des Motors setzen.

Zwei Wellenformen für die Kommutierung werden unterstützt, eine für 3-phasige Motoren mit 120 Grad Winkel zwischen den Phasen (z.B. bürstenlose Servomotoren) und eine für 2-phasige Motoren mit 90 Grad zwischen den Phasen (z.B. Schrittmotoren).

Andere Besonderheiten der POSYS® sind die Fähigkeit Hallsensoreingänge für die Phaseninitialisierung zu verwenden, einen Indextakt um die Kommutierungssynchronisation zu erhalten, den Enkodereingang vor zu skalieren um eine größere Anzahl verschiedene Arten von Rückmeldeeinheiten zu unterstützen und die Unterstützung von geschwindigkeits-basierter Phasenvorschau für weichere und effizientere Hochgeschwindigkeitsoperation.

¤

Auswahl vom 1-Phasen Signalausgabemodus mit der POSYS®

Auswahl vom 1-Phasen Signalausgabemodus mit der POSYS®

Um mit der POSYS® eine Achse im bürstenbehaftetem Modus zu betreiben, wird das Kommando SetNumberPhases verwendet. Zusätzlich sollte der Ausgabemodus entsprechend den Anforderungen wie im unteren Beispiel gezeigt, gesetzt werden. Das Kommando SetMotorType wird für die POSYS® 182x/192x Versionen verwendet.

POSYS® 8xx-B/9xx (Versionen für bürstenlose Servomotoren)

Kommando	Beschreibung
SetOutputMode	Ausgabemodus setzen
SetNumberPhases	Anzahl Phasen auf 1 setzen

POSYS® 182x/192x

Kommando	Beschreibung
SetMotorType	Ausgabemodus setzen

¤

Signalformen für die Kommutierung

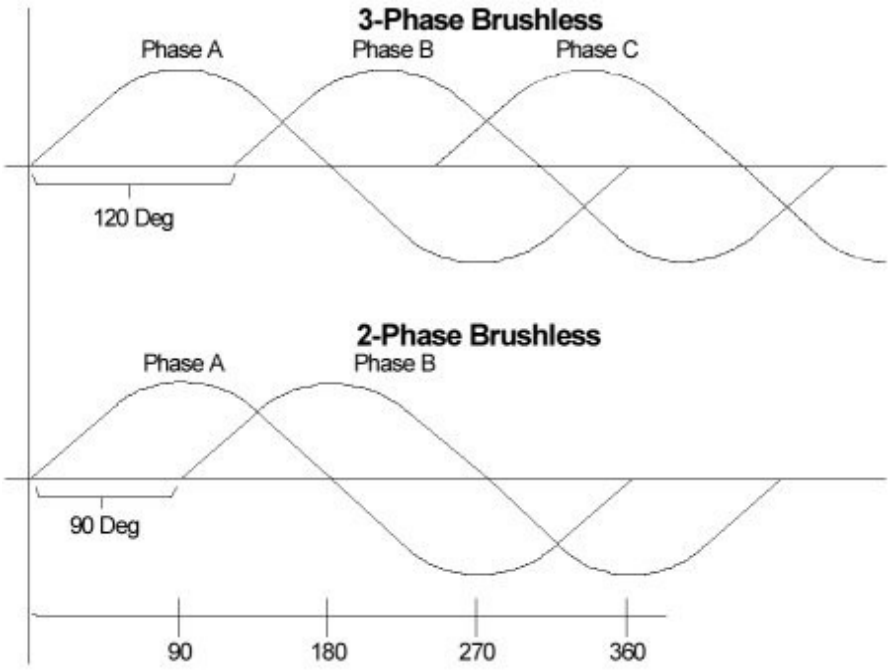
Signalformen für die Kommutierung

Die POSYS® unterstützt 2 verschiedene Signalformen für die Kommutierung. Die Eine ist eine um 120° versetzte Signalform, geeignet für 3-Phasen bürstenlose Servomotoren, die Andere eine um 90° versetzte Signalform, geeignet für 2-Phasen bürstenlose Motoren. Um die eine oder andere Signalform zu aktivieren, wird das Kommando SetNumberPhases n (2 oder 3; POSYS 8xx-B, 9xx) oder Set/GetMotorType (POSYS 18xx, 19xx) verwendet.

Abhängig von der Signalform und des gewünschten Motorausgangssignals (PWM oder DAC16), werden entweder 2 oder 3 kommutierte Ausgabesignale vom Controller pro Achse ausgegeben. Die folgende Tabelle listet die verschiedenen Kombinationen auf:

Signalform	Motorausgabemodus	Anzahl Ausgabesignale & Name
3-phase	PWM5050	3 (A, B, C)
3-phase	PWMSign/Mag	2 (A, B)
3-phase	DAC16	2 (A, B)
2-phase	PWM5050	2 (A, B)
2-phase	PWMSign/Mag	2 (A, B)
2-phase	DAC16	2 (A, B)

Das nachfolgende Diagramm zeigt die Phasen A, B und C für einen 3-Phasen bürstenlosen Motor und die A- und B-Phasen für einen 2-Phasen bürstenlosen Motor.



Set/GetNumberPhases, Set/GetMotorType

¤

Kommutierungsparameter

Kommutierungsparameter

Für die Sinuskommutierung ist es notwendig die Anzahl Enkoderstriche pro elektrischen Zyklus zu definieren. Um diese zu bestimmen müssen sowohl die Anzahl Enkoderauflösung pro Motorumdrehung als auch der Wert für die Anzahl elektrischer Zyklen des Motors bekannt sein. Wenn diese beiden Werte bekannt sind, kann die Anzahl Enkoderauflösungen pro elektrischen Zyklus mit folgender Gleichung ermittelt werden:

Counts_per_cycle = Counts_per_rot/Electrical_cycles

dabei ist:

Counts_per_rot ist die Enkoderauflösung pro Motorumdrehung

Electrical_cycles ist die Anzahl elektrischer Zyklen

Für die Anzahl elektrischer Zyklen verweisen wir hier auf die Spezifikationen des Motorenherstellers. Normalerweise ist die Anzahl elektrischer Zyklen identisch mit der Hälfte der Anzahl der Pole. Bitte verwechseln Sie nicht die Anzahl Pole mit der Anzahl Polpaare.

Das Kommando um die Anzahl Enkoderauflösung pro elektrischen Zyklus zu setzen ist SetPhaseCounts. Das Kommando GetPhaseCounts liest den programmierten Wert zurück.

¤

Indextakt bestimmen

Indextakt bestimmen

Um die Langzeitverlässlichkeit der Kommutierung zu verbessern, unterstützt die POSYS® die Möglichkeit einen Indextakt des Enkoders als Referenzpunkt während der Kommutierung zu verwenden. Durch Verwendung des Indextaktes während

der Phasenkalkulation wird jedweder Langzeitverlust an Enkoderzählungen, welche die Genauigkeit der Kommutierung beeinflussen können, eliminiert.

Um die Referenzierung des Indextaktes verwenden zu können, muss dieses Signal vom gewählten Enkodertyp zur Verfügung gestellt werden und zwar einmal pro Umdrehung.

Die Referenzierung des Indextaktes wird für alle bürstenlosen Motoren mit Inkrementalenkoder empfohlen. Für bürstenlose Linearmotoren wird er normalerweise nicht benötigt, obwohl er verwendet werden kann, solange der Indextakt so angelegt ist, dass jeder Takt im gleichen Phasenwinkel innerhalb des Kommutierungszyklus auftritt.

Die Verwendung des Indextaktsignals setzt nicht voraus, dass der Wert für die Enkoderauflösung pro elektrischen Zyklus ein Integer ist. Für den Fall es ist kein Integer sollte auf- oder abgerundet werden.

Umgekehrt, wenn kein Indextakt verwendet wird, dann muss die Enkoderauflösung pro elektrischen Zyklus exakt ein Integer ohne Rest sein.

z.B. bei Verwendung eines bürstenlosen Motors mit 6 Polen wäre ein Enkoder ohne Indextakt und 1200 Inkremente pro Umdrehung eine gute Wahl, aber ein Enkoder mit 1024 Inkremente wäre dies nicht, da 1024 nicht teilbar durch 3 ist ohne Rest.

Mit dem Kommando Set/GetPhaseCorrectionMode kann die Indextaktphasenkorrektur aktiviert bzw. deaktiviert werden.

Die Indextaktreferenzierung wird automatisch vom Controller ausgeführt, egal welcher Initializationsschema gewählt wurde (algorithmic, Hall-based, microstepping oder Direct Set).

¤

Kommutierungsfehlererkennung

Mit einem korrekt installierten Indextakt wird der Controller automatisch jedweden Verlust an Enkoderzählungen ausgleichen und korrigieren.

Jedoch sollte der Verlust größer werden, oder der Indextakt kommt nicht an innerhalb der erwarteten Position während des Kommutierungszyklusses, dann spricht man von einem „Kommutierungsfehler“. Der Bit 11 im Event Status Register für Kommutierungsfehler wird gesetzt.

Dieses Bit wird immer dann gesetzt falls die erforderliche Korrektur größer als (PhaseCounts/128)+4 ist.
Kommutierungsfehler können durch eine Vielzahl Umstände verursacht werden. Die Häufigsten sind im Folgenden gelistet:

- elektrische Störungen auf den A und B Enkoderleitungen
- elektrische Störungen auf der Leitung für den Indextakt
- falsches Setzen der Enkoderauflösung pro elektrischen Zyklus

Für jede Instanz in der ein Kommutierungsfehler auftritt wird die Phasenreferenzierung für den Indextakt ausgesetzt.

Abhängig vom Grund des Kommutierungsfehlers, könnte es ein einmaliges Ereignis sein oder kontinuierlich auftreten.

Sobald ein Kommutierungsfehler auftritt wird Bit #11 im Event Status Register „high“ gesetzt (1). Diese Bedingung kann auch als Quelle für die Verwendung von Hostinterrupts verwendet werden, um den Host automatisch über den aufgetretenen Kommutierungsfehler zu unterrichten. Dieses Bit muss vom Host zurückgesetzt werden, jedoch muss bedacht werden, dass je nach der Art des Fehlers der Kommutierungsfehler kontinuierlich auftreten kann.

Ein Kommutierungsfehler kann anzeigen, dass mit der Konfiguration des Positioniersystems ein ernsthaftes Problem besteht, welches sich in unsicheren Bewegungsbläufen äußert. Es liegt in der Verantwortung des Hosts/Bedieners den Fehler zu bestimmen und zu korrigieren um sichere Bewegungsabläufe zu gewährleisten.

¤

Phaseninitializierung

Nach dem Einschalten des Systems muss der Controller den korrekten Kommutierungswinkel relativ zur Enkoderposition bestimmen. Diese Information wird durch eine Prozedur, genannt Phaseninitialisierung, bestimmt und ermittelt.

Der Controller unterstützt 4 Methoden der Phaseninitialisierung; algorithmic, Hall Sensor-based, microstepping, und Direct-set.

Algorithmic Phase Initialization

Das Kommando SetPhaseInitializeMode mit dem Parameter algorithmic versetzt den Controller in diesen Modus.

Für diesen Modus benötigt der Motor keine anderen Informationen als die die von den Rückmeldeeinheiten zurückgeliefert werden. Um die Phasen zu bestimmen stimuliert der Controller kurz die Wicklungen des Motors und setzt die Phaseninitialisierung aufgrund der Motorreaktion. Aufgrund der resultierenden Bewegung kann der Controller automatisch die Phasen korrekt bestimmen.

Abhängig von der Größe und der Geschwindigkeit des Motors, kann die Phaseninitialisierung unterschiedlich lange dauern. Um diese Unterschiede anzupassen, kann die Wartezeit bis der Motor zum Stillstand kommt, mit dem Kommando SetPhaseInitializeTime programmiert werden. Das Kommando GetPhaseInitializeTime liest den programmierten Wert zurück.

Um die Wirkung auf die mechanischen Systemkomponenten so gering wie möglich zu halten, verwendet diese Methode einen Drehmomentsollwert, der vom Host gesetzt wird, um die Maximalmenge an Strom für die Phaseninitialisierung zu bestimmen. Als typische Werte sollte man von einem Wert der zwischen 5% und 25% des maximal benötigten Ausgangstromes liegt ausgehen, aber in der Regel 3-mal höher als der Anfahrwiderstand ist. Um bestmögliche Resultate zu erzielen, kann der Wert auch experimentell ermittelt werden. Das Kommando für den Motorbefehlswert ist SetMotorCommand. Der programmierte Wert kann mit dem Kommando GetMotorCommand zurückgelesen werden.

Um die Initialisierungsprozedur auszulösen, verwendet man das Kommando InitializePhase. Dieses Kommando wird sofort ausgeführt, ein Update-Kommando ist nicht notwendig.

Jedoch, ehe das Kommando für die Phaseninitializierung ausgeführt wird (InitializePhase), muss der Motor ausgeschaltet (SetMotorMode), ein Drehmomentsollwert definiert (SetMotorCommand) und die Zeitspanne für die Initialisierung bestimmt sein (SetPhaseInitializeTime).

Während der Phaseninitialisierung nach der Algorithmusmethode kann sich der Motor plötzlich in die eine oder andere Richtung bewegen.

Es müssen Vorsichtsmassnahmen ergriffen werden um Schäden zu vermeiden. Zusätzlich, um akkurate Ergebnisse zu erzielen, muss die Bewegung in beide Richtungen ungehindert ablaufen können und darf nicht durch übermäßige Anfahrreibung behindert werden.

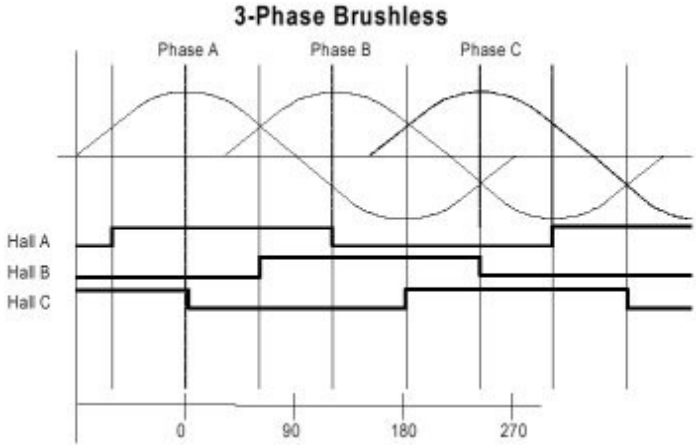
Hall-Based Phase Initialization

Um den Controller für die Hallsensor-basierte Initialisierung zu verwenden, wird das Kommando SetPhaseInitializeMode und der Parameter Hall-basiert verwendet.

In diesem Modus werden 3 Hallsensorsignale verwendet um die Phasen des Motors zu bestimmen und die Sinuskommutierung beginnt automatisch nachdem der Motor eine volle Umdrehung absolviert hat.

Die Anschlüsse für die Hallsensoren werden über den optionalen Stecker mit den Kennzeichnungen Hall1A-C (Achse #1) und Hall2A-C (Achse #2) usw. zurückgeführt. Es sollte Sorgfalt angewandt werden beim Anschliessen der Hallsensoren. Um den gegenwärtigen Status der Hallsensoren zu erfahren, kann das Kommando GetSignalStatus verwendet werden.

Das folgende Diagramm zeigt die Beziehung zwischen den 3 Hallsensoreingängen jeder Achse und den kommutierten Ausgangssignalen. Dieses Diagramm zeigt die erwarteten Hall-Sensorenzustände und Wicklungsanregung für die Vorwärtsbewegung.



Im Gegensatz zu der algorythmischen Methode, sind bei der Hallsensor-basierten Phaseninitialisierung keine besonderen Motoreinstellprozeduren notwendig. Die Initialisierung wird ausgeführt mit dem Kommando InitializePhase, und wird sofort ohne eine Motorbewegung aktiv.

Um sich den verschiedenen Typen von Hallsensoren anzupassen, auch jenen mit invertierter Schaltung, kann die Signalinterpretation der Signallogik vom Host gesetzt werden.

Das Kommando SetSignalSense akzeptiert ein Bit-programmiertes Wort das überprüft ob die ankommenden Hall-Signale als aktiv "high" oder aktiv "low" interpretiert werden. Um den programmierten Hall-Interpretationswert zurückzulesen wird das Kommando GetSignalSense verwendet.

Die Hall-basierte Initialisierung sollte nur mit der 3-Phasen Kommutierungsmethode verwendet und mit Hallsensoren die um 120° versetzt angelegt sind. Hallsensoren die nur um 60° versetzt angelegt sind, sollten nicht verwendet werden.

Microstepping Phase Initialization

Falls die Position des Indexsignals in Relation zum Motorrotor und Gehäuse bekannt ist, kann es vorteilhaft sein eine Initialisierungstechnik zu verwenden die den Motor als Mikroschrittmotor anspricht, den Motor in eine Richtung bewegen bis das Indexsignal gefunden wurde und anschliessend die Phasenwinkel entsprechend setzen.

Dieses System ist nur für Motoren geeignet die das Indexsignal an einer fixen und wiederholbaren Position innerhalb des Kommutierungszyklus haben, die für alle Motoren gleich ist. Obwohl dies relativ ungewöhnlich ist, ist es typisch für Motoren mit optischen Hallsensoren die eine einzelne Scheibe verwenden die A, B, das Indexsignal und Hallsensorinformation gleichermaßen enthält.

Um die Phaseninitialisierung für den Mikroschrittmodus zu verwenden, verwendet man das Kommando SetCommutationMode Microstepping. Um den Controller in den Modus für die enkoder-basierte Kommutierung zurück zu versetzen, verwendet man das Kommando SetCommutationMode Sinusoidal. Sobald das Indexsignal erfasst wurde, kann mit dem Kommando SetPhaseAngle der Phasenwinkel gesetzt werden.

Um den Motor im Mikroschrittmodus verwenden zu können, muss der Motor eingeschaltet (SetMotorMode) und ein Drehmomentsollwert für den Motor definiert sein (SetMotorCommand). Zusätzlich sollte die Enkoderauflösung pro elektrischen Zyklus auf 512 eingestellt sein.
Denn im Mikroschrittmodus zählt jeder Trajektoriezähler 1/256 eines vollen elektrischen Zyklus.

Z.B. mit einem 4-poligen Motor (2 elektrische Zyklen pro Umdrehung) würde eine Trajektoriebewegung mit 512 Zählern den Motor eine volle Umdrehung machen lassen.

Besondere Vosichtsmassnahmen sollten ergriffen werden, wenn der Motor im Mikroschrittmodus verwendet wird. Weil der Motor im "Open-Loop-Modus" verwendet wird, könnte die resultierende Spulenerregung und darauffolgende Rotation ruckartig und abrupt sein.

Nur unter besonderen Umständen sollte die Phaseninitialisierung im Mikroschrittmodus erfolgen.
Es ist nicht sonderlich angeraten, außer wenn die algorithmische oder Hall-basierte Methode nicht verwendet werden können.

Direct-Set Phase Initialization

Falls nach dem Einschalten die Phasen bekannt sind, können diese explizit mit dem Kommando SetPhaseAngle gesetzt werden.

Dies kommt typischerweise dann vor, wenn Sensoren wie Resolver benutzt werden, bei der zurückgelieferten Motorpositionsinformationen absolut sind (nicht inkremental), und benutzt werden können um ein Rechteckdatensignal zu generieren als auch vom Hostrechner direkt gelesen werden können.

¤

Programmierung der Phaseninitializierung

Die folgenden Beispiele zeigen typische Kommandofolgen eines bürstenlosen Motors für alle vier Initialisierungsmethoden, um die Kommutierung zu initialisieren.

Algorithmic Initializierungssequenz

Kommando	Beschreibung
SetOutputMode m	Motorausgabemodus setzen
SetNumberPhases p	Anzahl der Phasen setzen
SetPhaseCounts uuuu	Anzahl der Encoderstriche pro elektrischen Zyklus setzen
SetPhaseInitializeMode Algorithmic	Algorithmische Initialisierung setzen
SetMotorMode Off	Motor ausschalten damit es nicht mit der Initialisierungsprozedur in Konflikt gerät.
SetPhaseInitializeTime wwww	Initialisierungsdauer der Phaseninitialisierung setzen.
SetMotorCommand yyyy	Einstellen des Initialisierungsmotorausgabesignals.
InitializePhase	Initialisierung starten.

Diese Sequenz veranlasst den Motor sofort mit der Initialisierung zu beginnen die "wwww" Anzahl von Servoloops andauert. Um zu bestimmen ob die Prozedur komplettiert wurde, kann man mit dem Kommando GetActivityStatus überprüft werden. Der "Phase Initialization" Bit wird anzeigen ob die Prozedur beendet wurde. Nach dem der Initialisierungsprozess beendet wurde, sollte der Motor mit dem Kommando SetMotorMode On eingeschaltet werden, falls der Motor im Closed-Loop-Mode betrieben werden soll.

Hall-based Initializierungssequenz

Kommando	Beschreibung
SetOutputMode m	Motorausgabemodus setzen
SetNumberPhases p	Anzahl der Phasen setzen
SetPhaseCounts uuuu	Encoderauflösung pro elektrischen Zyklus setzen
SetSignalSense vvvv	Interpretation der Signalerkennung auf Hallsensorerkennung setzen
SetPhaseInitializeMode	Phaseninitialisierungsmethode auf Hallsensor-basiert setzen
InitializePhase	Initialisierung starten

Diese Sequenz setzt den Controller in den Modus die Hallsensor-basierte Initialisierung anzuwenden und die Phaseninitialisierung sofort zu beginnen. Der Motor wird sich nicht bewegen aufgrund dieser Sequenz und weitere Operationen werden nicht ausgeführt.

Microstepping Initializierungssequenz

Kommando	Beschreibung
SetOutputMode m	Motorausgabemodus setzen
SetNumberPhases p	Anzahl der Phasen setzen

Kommando	Beschreibung
SetPhaseCounts 512	Encoderauflösung pro elektrischen Zyklus setzen
SetCommutationMode Microstepping	Setzen des Controllers in den Kommutierungsmodus “Microstepping”
SetCaptureSource Index	Setzen des “capture mode” zu Index (nicht notwendig falls schon geschehen).
ResetEventStatus 0	Achsenstatus auf Null setzen
GetCaptureValue	Alle vorhergehenden Capturewerte auf Null setzen
SetMotorMode On	Motor einschalten (nicht notwendig falls schon geschehen).
SetMotorCommand xxxx	Wert für die Motorausgabe setzen
SetPosition 560	Verfahrstrecke definieren; sollte etwas mehr als eine volle Umdrehu betragen (vorausgesetzt 4-poliger Motor).
SetVelocity yyyy	Geschwindigkeit setzen
SetAcceleration zzzz	Beschleunigung setzen
Update	Werte aktivieren und Initialisierung starten

Diese Sequenz veranlasst den Motor etwas mehr als eine volle Umdrehung zu machen. Nach dem Update sollte der Host das Statuswort (GetEventStatus) pollen bis eine Erfassung erfolgt und dann sofort ein SetPhaseAngle Kommando schicken, gefolgt von einem SetPhaseOffset Kommando, beides mit den Werten für die Phasenwinkel für die Phaseninitialisierung.

Mehr Informationen bezüglich der Bestimmung der korrekten Werte für die Phasen erhalten Sie im Kapitel "Justierung der Phasenwinkel".

Nachdem die Kommandos SetPhaseAngle und SetPhaseOffset vom Host geschickt wurden, sollte der Controller für die normale Kommutierungsoperation initialisiert werden. Dies bedeutet der Phasenmodus sollte auf Enkoder-basiert (SetCommutationMode Sinusoidal) und die korrekte Enkoderauflösung pro elektrischer Zyklus sollte mit dem Kommando SetPhaseCounts eingestellt sein.

Direct-Set Initializierungssequenz

Kommando	Beschreibung
SetOutputMode m	Motorausgabemodus setzen
SetNumberPhases p	Anzahl der Phasen setzen
SetPhaseCounts xxxx	Encoderauflösung pro elektrischen Zyklus setzen (hex).
SetPhaseAngle yyyy	Phasenwinkel setzen, basierend auf Informationen vom externen Sensor.

Diese Sequenz wird den Phasenwinkel direkt setzen, der von einem externen Sensor bestimmt wurde. Der Wert muss zwischen 0 und der maximalen Enkoderauflösung pro elektrischer Zyklus sein.

¤

Justierung der Phasenwinkel

Justierung der Phasenwinkel

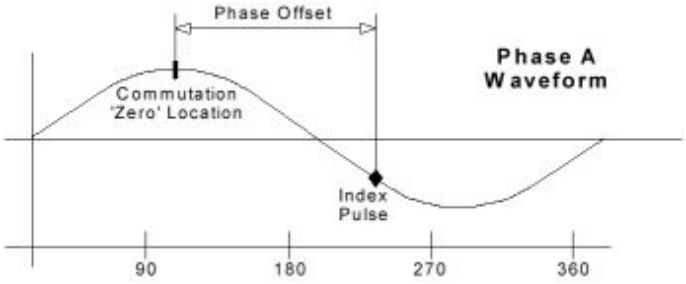
Die POSYS® für bürstenlose Servomotoren verfügen über die Fähigkeit den Kommutierungswinkel der Motor direkt zu ändern, im Stillstand als auch während der Bewegung. Obwohl dies nicht unbedingt notwendig ist kann es zum Testen ganz nützlich sein, oder während der Kommutierungsinitialisierung im Mikroschritt bzw. Direkt-Set Modus.

Um den Kommutierungswinkel zu ändern, wenn der Motor stillsteht, wird das Kommando SetPhaseAngle verwendet.

Um den Kommutierungswinkel zu ändern, während sich der Motor bewegt, ist auch der Indextakt notwendig und das Kommando SetPhaseOffset wird verwendet, welches nur dann ausgeführt wird, wenn ein Indextakt erfasst wird. Die folgende Beschreibung liefert einiges Hintergrundwissen zu dieser Funktion.

Nach der Phaseninitialisierung wird der korrekte Kommutierungswinkel vom Controller als Relativwert vom Index (in Inkrementen) zum maximalen Ausgabewert der Phase A gespeichert (commutation "zero" location). Dieser 16-Bit Registerwert kann mit dem Kommando GetPhaseOffset gelesen werden.

Die folgende Grafik zeigt die Abhängigkeit zwischen der Kommutierungsnullposition der Phase A, der Indexposition und dem Offsetwert der Phase. Der gezeigte Indextakt für einen gegebenen Motor in dieser Grafik könnte irgendwo innerhalb eines Phasenzyklusses lokalisiert worden sein, da es von Motor zu Motor variieren kann. Nur bei Motoren die dahingehend montiert wurden, dass die Indexposition immer im gleichen Bezug zu den Motorwicklungen stehen, werden eine konsistente Indexposition relativ zur Kommutierungsnullposition aufweisen.



Der Phasenoffsetregister wird vor der Phaseninitialisierung den Wert ffff (hex) aufweisen. Nach der Phaseninitialisierung und nachdem der Motor mindestens eine Umdrehung dahingehend vollführt hat, dass ein Indextakt erfasst werden konnte, wird der Phasenoffsetwert als positive Nummer mit einem Wert zwischen 0 und der Enkoderauflösung pro elektrischen Zyklus gespeichert.
Um den Phasenoffsetwert von einer Nummer in Grad zu konvertieren, kann folgender Ausdruck verwendet werden:

Offsetdegrees = 360 * Offsetcounts/counts_per_cycle

wo:

Offsetdegrees	//ist der Phasenoffset in Grad
Offsetcounts	//ist der Phasenoffset in Enkoderstriche
counts_per_cycle	//ist die Anzahl Enkoderstriche pro elekctrischen Zyklus die mit //dem Kommando SetPhaseCounts gesetzt wurde.

Der Phasenoffsetwert kann, während der Motor in Bewegung ist, beliebig oft verändert werden, wobei nur kleine Änderungen vorgenommen werden sollten um plötzliche, große Sprünge zu vermeiden.

Die Kommandos SetPhaseOffset und GetPhaseOffset können nur verwendet werden, wenn ein Enkoder mit Indextakt angeschlossen ist. Wenn der Indextakt nicht angeschlossen oder verwendet wird, kann der Phasenoffsetwinkel nicht justiert bzw. zurückgelesen werden.

Die relative Phasenpositionierung von B und C zu A werden durch das Setzen von Phasenoffsetwerten nicht verändert. Diese Phasen bleiben bei 90° bzw. 120° Offset zu Phase A (abhängig von der gewählten Signalform).

¤

Encoder Pre-Scalar

Encoder Pre-Scalar

Besonders in Verbindung mit Linearmotoren kann die Enkoderauflösung pro elektrischen Zyklus stark variieren. Typischerweise haben Motoren Enkoderzählwerte zwischen 1 und 32767. Lineare bürstenlose Motoren dagegen können Werte von 1000000 pro Zyklus und höher haben wegen der öfteren Verwendung von laser-basierten Enkoder mit höchster Genauigkeit.

Um diesen großen Bereich abzudecken, verfügen die POSYS® für bürstenlose Servomotoren über eine Preskalarfunktion welches, für den Zweck der Kommutierungskalkulationen, die eingehenden Inkremente durch 64 teilt. Mit der aktivierten Preskalarfunktion kann der maximale Bereich von Inkrementen pro elektrischer Zyklus dann 2097088 betragen.

Das Kommando SetPhasePrescale On aktiviert den Modus. Das Kommando SetPhasePrescale Off deaktiviert ihn.

Die Preskalarfunktion hat nur auf die Kommutierung des Controllers Einfluss. Es beeinflusst nicht die Position durch die Servoregelung oder abgefragt durch das Kommando GetActualPosition.

Die Preskalarfunktion sollte weder aktiviert noch deaktiviert werden wenn der Motor sich in Bewegung befindet.

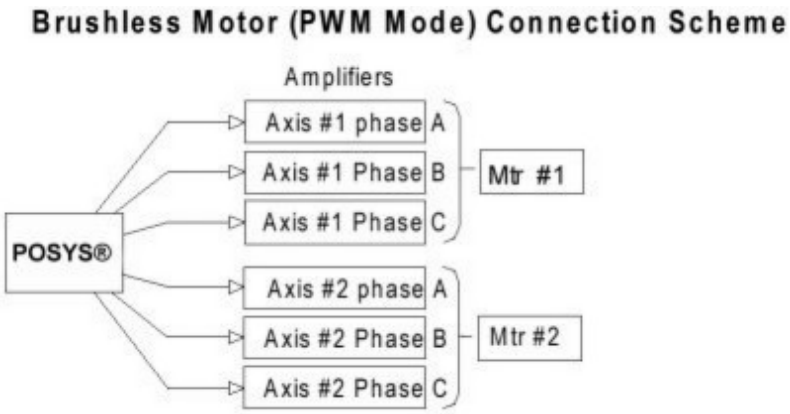
¤

Konfiguration der Motorsignalausgabe

Konfiguration der Motorsignalausgabe

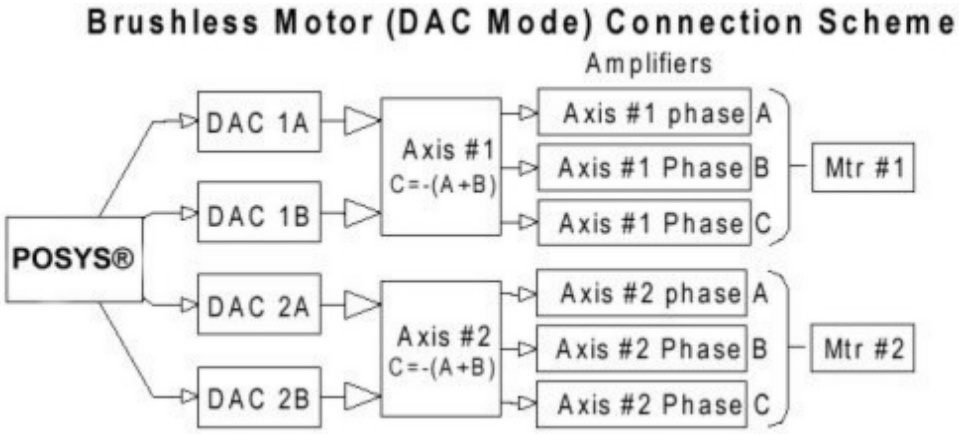
Die POSYS® für bürstenlose Servomotoren unterstützen 2 Arten von Motorausgabemethoden, PWM und DAC (bis zu 16 Bit Auflösung)(Set/GetOutputMode).

Nachfolgend ist eine typische Verstärkerkonfiguration für ein 3-Phasen bürstenloser Servomotor im PWM-Modus:



In dieser Konfiguration gibt der Controller 3-phasige impuls-breiten-modulierte Signale pro Achse aus. Diese Signale gehen direkt auf 3 half-bridge-artige Verstärker.

Nachfolgend ist eine typische Verstärkerkonfiguration für ein 3-Phasen bürstenloser Servomotor im DAC-Modus:

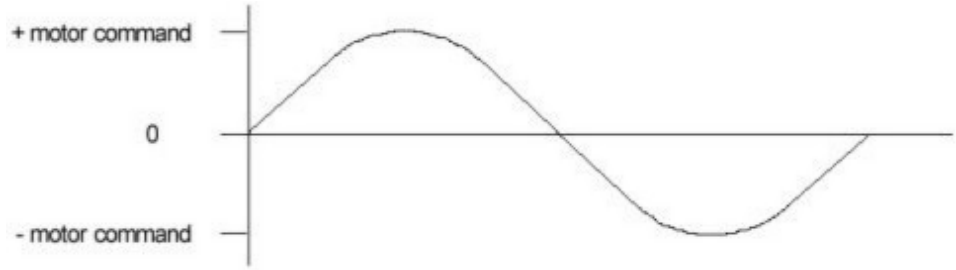


Bei der Verwendung des DAC-Ausgabemodus muss der Digitalwert des Controllers in eine Spannung konvertiert werden. Dafür wird ein externer DAC verwendet. Zwei DAC Kanäle sind pro Achse notwendig. Um den Wert für die dritte Phase für einen bürstenlosen Servomotor (C-Phase) zu konstruieren, wird folgende Formel verwendet (C = - (A+B)).

Dies wird gewöhnlich durch einen Operationsverstärkerschaltkreis bewerkstelligt. Zusätzlich, falls der Drehmomentsollwertmodus gewünscht wird, ist die Summe der Ströme der 3 Ausgangssignale die durch die Wicklungen des Motors fliessen gleich 0.

Interpretation des Motorausgangssignals

Das folgende Diagramm zeigt das gewünschte Ausgangspannungsliniendiagramm für eine einzelne Phase.



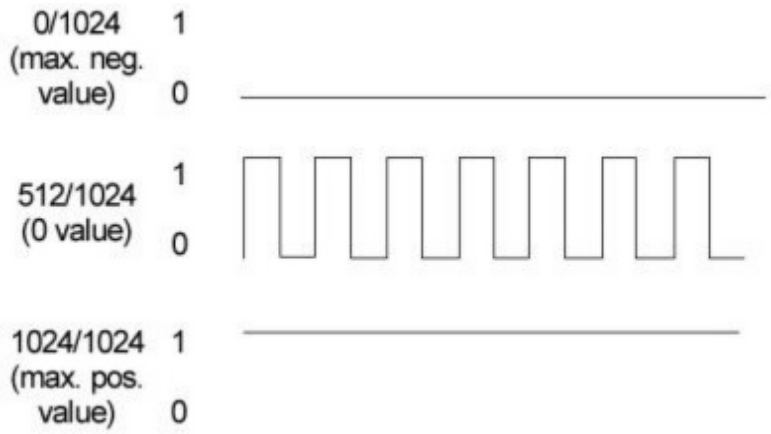
Das Liniendiagramm ist um einen Wert von 0 Volt zentriert. Die Magnitude des generierten Liniendiagrammes ist proportional zur Ausgabe des Servofilters oder des Motorbefehlsregisters (abhängig vom Kommutierungsmodus und ob der Motor an oder aus ist).

Zum Beispiel wenn der Regler mit einem DAC mit Ausgangsbereich -10V bis +10V verbunden wird und der Regler ist im Open-Loop-Modus mit einem Motorausgabewert von 32767 (Maximalwert) dann, wie der Motor durch eine volle elektrische Periode rotiert, wird ein sinusförmiges Signal bei 0 Volt zentriert mit einem minimum Wert von -10V und einem maximum Wert von +10V ausgegeben.

PWM Decoding

Das Ausgangssignal für den PWM Modus erlaubt auch eine sinusförmige Signalform für jede Phase; jedoch die Methode mit der diese Signale die Spannung codieren unterscheiden sich grundsätzlich vom DAC16 Digitalwort. Das PWM Ausgangssignal verwendet ein einzelnes Signal pro Motorausgangsphase. Dieses Signal beinhaltet eine impuls-breiten kodierte Representation der gewünschten Spannung. Der Arbeitszyklus der Signalform bestimmt die Kodierung der gewünschten Spannung. Die Frequenz des PWM Zylus ist 20 kHz, mit einer Auflösung von 10 Bit, oder 1/1024 bzw. alternativ auch 80 kHz mit einer Auflösung von 8 Bit bei den neuen POSYS 182x/192x.

Die folgende Tabelle zeigt das Schema:



Ein Ausgabetakt mit 0 Anteilen pro 1024 möglichen Anteilen representiert die maximale negative Spannung. Ein Ausgabetakt von 512 pro 1024 (50%) möglichen Anteilen representiert eine Ausgabespannung von 0V und ein Ausgabetakt von 1024 pro 1024 möglichen Anteilen representiert die maximale positive Spannung.

Set/GetNumberPhases, Set/GetMotorType

¤

Open Loop Stepper Control

Überblick

Überblick

Dieses Kapitel beschreibt die Open-Loop Schrittmotor Funktionalitäten der POSYS® im Schrittmotormodus bzw. der reinen Schrittmotorkarte.
Alle dieser Controller verfügen über Enkoderanschlussmöglichkeiten die dafür benutzt werden können, einen Stillstand in der Bewegung des befestigten Motors zu entdecken und außerdem über eine für die Schrittmotorkontrolle konstruierte Schaltungsanordnung verfügen.

¤

Trajectory Control Units

Profilberechnung

Für die POSYS® Schrittmotorprodukte gelten folgende Unterschiede, die POSYS® in Schrittmotormodus gibt Takt & Richtungssignale aus, die POSYS® in Mikroschrittmodus gibt Mikroschritte aus, dagegen sind in den Servoprodukten alle Einheiten Enkodereinheiten. Die nachfolgende Tabelle zeigt die Kommandos und die entsprechenden Einheiten.

Kommando	Servo	Mikroschritt	Schritt
Set/GetPosition	counts	micro-steps	steps
Set/GetVelocity	counts/cycle	micro-steps/cycle	steps/cycle
Set/GetAcceleration	counts/cycle²	micro-steps/cycle²	steps/cycle²
Set/GetDeceleration	counts/cycle²	micro-steps/cycle²	steps/cycle²
Set/GetJerk	counts/cycle³	micro-steps/cycle³	steps/cycle³
Set/GetStartVelocity	-	micro-steps/cycle	steps/cycle
GetCommandedPosition	counts	micro-steps	Steps
GetCommandedVelocity	counts/cycle	micro-steps/cycle	steps/cycle
GetCommandedAcceleration	counts/cycle²	micro-steps/cycle²	steps/cycle²
Set/GetPositionErrorLimit	counts	micro-steps	Steps
GetPositionError	counts	micro-steps	steps

¤

Enkoderrückmeldung

Enkoderrückmeldung

Alle POSYS® unterstützen inkrementale Enkoderrückmeldung, optional auch Absolutenkoder mit entsprechender Zusatzhardware (z.B. POSYS® SSI 800/900). Nach dem Einschalten oder nach einem Controller-Reset wird die Enkoderrückmeldung (GetEncoderSource) auf Null gesetzt, welches die Enkoderrückmeldung insgesamt optional macht. In diesem Modus wird die Enkoderposition ignoriert. Die aktuelle Position kann mit dem Kommando GetActualPosition abgefragt werden.

Das Kommando SetActualPosition kann verwendet werden um die aktuelle Position zum programmierten Wert zu machen. Die Einheiten für dieses Kommando sind Inkremente. Um das Programmdesign und -debugging zu vereinfachen, können aktuelle Positionseinheiten in Schritt/Mikroschritte geändert werden. Dafür gibt es das Kommando SetActualPositionUnits. Die Tabelle zeigt die Kommandos die dies beeinflussen:

Kommando	Positionseinheiten = counts	Positionseinheiten = steps
Set/GetActualPosition	counts	steps/micro-steps

Kommando	Positionseinheiten = counts	Positionseinheiten = steps
AdjustActualPosition	counts	steps/micro-steps
GetCaptureValue	counts	steps/micro-steps

¤

Stall Detection

Stall Detection

Zusätzlich zur passiven Positionserfassung durch den Host mit dem GetActualPosition Kommando, kann die POSYS® auch aktiv das Ziel und die aktuelle Position überwachen und einen Bewegungsfehler entdecken, der einen Stillstand zum Ergebnis hat. Die automatische Stillstandserfassung erlaubt den Controller festzustellen wann der Schrittmotor während der Bewegung Schritte verloren hat. Gewöhnlich passiert dies wenn der Motor auf ein Hindernis trifft oder wenn auf andere Art und Weise sein Nennmoment überschritten wird.

Die automatische Stillstandserfassung arbeitet nach der Initialisierung kontinuierlich. Die Sollposition wird mit der Istposition (Enkoder) verglichen und falls der Unterschied dieser beiden Werte einen vorher definierten Wert überschreitet wird ein sogenannter "Stall" erkannt. Im anwenderprogrammierten Register SetPositionErrorLimit wird der Schwellenwert bestimmt für den Bewegungsfehler (Stall).

Um die automatische Bewegungsfehlererkennung zu initiieren, muss vom Host aus die Anzahl Inkremente pro Schritt/Mikroschritt spezifiziert werden. Dies wird mit dem Kommando SetEncoderToStepRatio erfüllt. Dieses Kommando akzeptiert 2 Parameter. Der 1. Parameter ist die Anzahl Inkremente pro Motorumdrehung und der 2. Parameter ist die Anzahl Schritte/Mikroschritte pro Motorumdrehung.

Parameter	Format	Wortgröße	Bereich
Encoder counts per rev	16.0	16 Bit	0 bis 32767
Steps/micro-steps per rev	16.0	16 Bit	0 bis 32767

Z.B. falls ein Schrittmotor mit 1,8° pro Vollschritt verwendet wird mit einem Enkoder mit 4000 Inkremente pro Umdrehung, müssten die Parameter wie folgt aussehen:

SetEncoderToStepRatio 4000 200

wobei die Anzahl Schritte pro Umdrehung sich wie folgt errechnet: 360/1,8.

Falls der gleiche Motor mit den Mikroschrittversionen der POSYS® verwendet wird und die Anzahl Mikroschritte pro Vollschritt ist auf 64 gesetzt, dann sehen die Parameter wie folgt aus:

SetEncoderToStepRatio 4000 12800

wobei die Anzahl Schritte pro Umdrehung wie folgt errechnet wird (360/1,8)*64.

In jenen Fällen in denen die Anzahl Schritte, Mikroschritte oder Inkremente pro Umdrehung das Maximum von 32767 überschreitet, können die Parameter als Bruchteile einer Umdrehung spezifiziert werden, solange das Verhältnis genau bewahrt wird. Von dem vorhergehenden Beispiel ausgehend kann das Verhältnis auch nachfolgend dargestellt werden:

SetEncoderToStepRatio 2000 6400

dabei das Verhältnis für eine halbe Umdrehung anzeigend. Das Verhältnis für einen Bruchteil einer Umdrehung anzugeben ist genau so genau, wie man es für eine volle Umdrehung angibt.

Eine typische Sequenz für Bewegungsfehlererkennung wird unten gezeigt.

Kommando	Beschreibung
SetEncoderSourceIncremental	Art der Enkoderrückmeldung setzen
SetEncoderToStepRatio4096 200	Setzen des Verhältnisses von Schritten/Mikroschritten zu Inkrementen pro Umdrehung. Setzen des Fehlerfensters in Einheiten von Schritten/Mikroschritten
SetPositionErrorLimit	Setzen des Fehlerfensters in Einheiten von Schritten/Mikroschritt

Kommando	Beschreibung
ClearPositionError	Setzen des Fehlerfensters in Einheiten von Schritten/Mikroschritt
Update	Senden der gebufferten Kommandos an den Motionprozessor
SetAutoStopModeOn	Ermöglicht das Anhalten des Motors wenn ein Bewegungsfehler erkannt wurde

Sobald ein Bewegungsfehler auftritt, treten weitere Ereignisse gleichzeitig auf. Der Motion Error Bit des Event Status Word wird gesetzt. Falls der "automatische Stopp bei Bewegungsfehler" aktiviert ist, wird der Motor ausgeschaltet, welches den Effekt hat dass der Trajektoriegenerator deaktiviert wird.

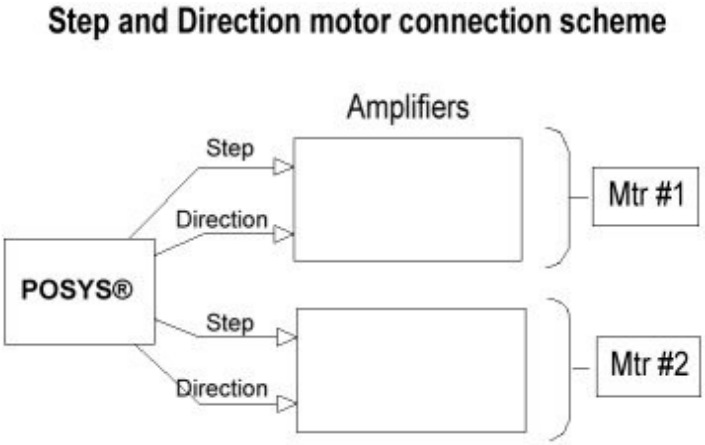
¤

Takt & Richtungssignalgenerierung

Takt & Richtungssignalgenerierung

Für jede Achse werden 2 Signale ausgegeben um die gewünschte Position zu erhalten zu einem bestimmten Moment. Diese beiden Signale sind das Taktsignal und das Richtungssignal.

Das Taktsignal, dass vom Controller ausgegeben wird, besteht aus einer kontrollierten Serie individueller Takte die jedes für sich einen gewünschten Schritt repräsentieren. Dieses Signal wird immer als Rechtecksignal ausgegeben. Als Standard gilt ein Taktsignal als gegeben wenn das Signal von "high" auf "low" geht. Eine Invertierung dieser Signale wird später in diesem Kapitel beschrieben. Das Richtungssignal wird mit dem Taktsignal synchronisiert ausgegeben. Das Richtungssignal ist dahingehend kodiert, dass ein "high" für die positive und ein "low" für die negative Richtung gilt.



Die POSYS® für Schrittmotoren unterstützen verschiedene Taktraten, die mit dem Kommando SetStepRange eingestellt werden. Die folgende Tabelle listet die verschiedenen Möglichkeiten auf:

Kommando	Frequenzbereich der Taktsignale
SetStepRange 1	0 to 4.98 M Schritte pro Sekunde
SetStepRange 4	0 to 622.5 K Schritte pro Sekunde
SetStepRange 6	0 to 155.625 K Schritte pro Sekunde
SetStepRange 8	0 to 38.90625 K Schritte pro Sekunde

Die Bereiche oben zeigen die minimalen und die maximalen Bereiche die auf dem Controller eingestellt werden können. Z.B. falls die gewünschte maximale Schrittrate 200K pro Sekunde ist, dann ist die korrekte Einstellung SetStepRange 4.

Für Voll- und Halbschrittapplikationen, sowohl auch Takt- und Richtungsapplikationen die eine maximale Schrittrate von 38KSchritt/Sek. haben, sollte SetStepRange 8 verwendet werden. Für Applikationen die eine höhere Schrittrate als 38 KSchritt/Sek. erfordern, sollten höhere Einstellungen verwendet werden.

Für jede Achse kann eine individuelle Einstellung gewählt werden. Um die gewählte Einstellung abzufragen kann das Kommando GetStepRange verwendet werden.

Der Taktzähler ist standardmäßig so eingestellt, dass ein zu machender Schritt als solcher erkannt wird wenn das Signal von "high" auf "low" geht. Manche Treiber benötigen aber das Signal ganau anders herum. Mit dem Kommando SetSignalSense lässt sich die Logik umdrehen. In der Beschreibung des Kommandos SetSignalSense kann man hierüber mehr erfahren.

¤

Pulse Generation Control

Pulse Generation Control

Die Rate der Taktsignale wird üblicherweise von den besonderen Profilparametern bestimmt, die vom Host verlangt werden. Zusätzlich zum Bahnprofil jedoch gibt es eine gesonderte Methode die Taktgenerierung zu aktivieren bzw. deaktivieren. Diese Methode ist unter der Bezeichnung 'Motor-Kontrolle' bekannt und unterstützt einen EIN/AUS Taktgeneratorsteuermechanismus.

Um die Taktausgabe zu aktivieren, verwendet man das Kommando SetMotorMode ON und um sie zu deaktivieren das Kommando SetMotorMode OFF. SetMotorMode OFF veranlasst den Profilgenerator sofort die Taktausgabe zu unterbrechen bis ein SetMotorMode ON Kommando erteilt wird. Alle anderen Bahn- oder Profilkommandos werden nicht ausgeführt, bis ein SetMotorMode ON Kommando den Motor wieder aktiviert.

Mit dem Kommando GetMotorMode kann man den Status (AN oder AUS) abfragen.

Wenn der Motor vom Host eingeschaltet wird (Kommando SetMotorMode ON) wird er in der Stillstandsposition verbleiben bis ein neuer Profil geladen und initiiert wurde. Falls ein Profil vorher schon programmiert und geladen wurde wird es ignoriert.

¤

AtRest Indikator

AtRest Indikator

Zusätzlich zu den Takt- & Richtungssignalen unterstützt die POSYS® für Schrittmotoren einen weiteren Ausgang für jede Achse, bekannt als das AtRest Signal, welches anzeigt ob ein Profil gerade abgearbeitet wird oder nicht. Dieses Signal kann hilfreich sein in Verbindung mit Treibern die ein anderes Moment im Stillstand aufweisen als während der Bewegung.

Diese Funktion steht jederzeit zur Verfügung und muss nicht durch den Host aktiviert werden.

¤

Mikroschritt Sinus/Cosinus Motorausgangssignal

DAC und PWM Motorausgabe (POSYS® Mikroschrittmodus)

DAC und PWM Motorausgabe (POSYS® Mikroschrittmodus)

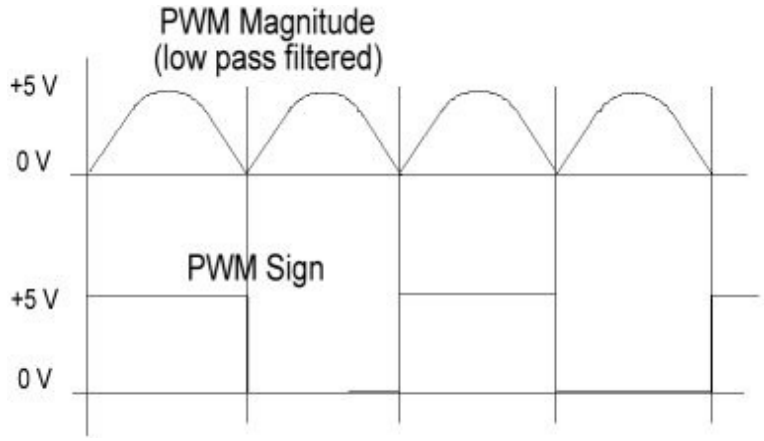
Die POSYS® unterstützen 2 verschiedene Arten von Motorausgangssignalen, PWM und DAC. Die Ausgabe wird vom Host definiert und ist pro Achse mit dem Kommando SetMotorCommand mit entsprechendem Parameter einstellbar. Eine 0 setzt die Ausgabe auf DAC und eine 1 auf PWM (sign/magnitude).

¤

Motorsignalausgabe Interpretation

Motorsignalausgabe Interpretation

Das folgende Diagramm zeigt die typische Signalform eines Signals einer einzelnen Phase für die Mikroschrittversionen. Jede Phase gibt ein ähnliches wellenförmiges Signal aus, wobei die Signale um 90° bzw. 120° verschoben sind (je nachdem ob für 2-Phasen oder für 3-Phasen Schrittmotoren.



Die Ausgabe der Kurve ist um den Nullwert zentriert. Die Magnitude der gesamten Signalform wird vom Motor Command Register (SetMotorCommand) kontrolliert.

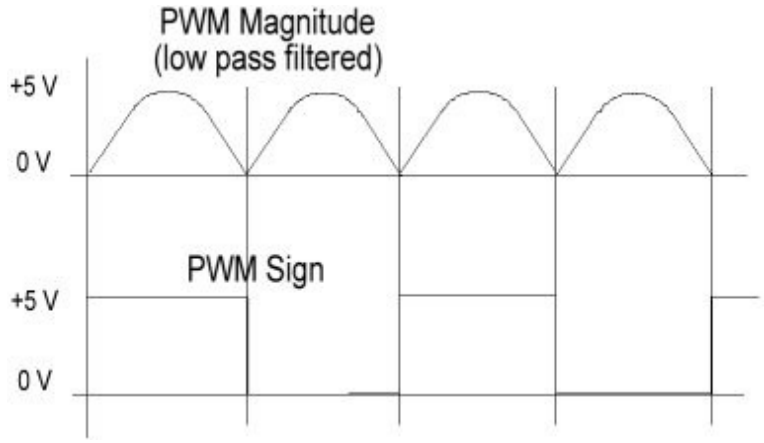
Z.B. wenn die Ausgabe über einen DAC erfolgt mit min/max. von -10V bis +10V und der maximale Motorausgabewert ist mit dem Kommando SetMotorCommand auf 32767 gesetzt, dann, während der Motor eine Umdrehung vollführt, wird die Sinuskurve einmal um die 0V zentriert eine minimale Spannung von -10V und eine maximale Spannung von +10V ausgeben.

¤

PMW Dekodierung

PWM Dekodierung

Die PWM-Methode gibt auch eine sinusförmige Spannung für jede Phase aus, jedoch die Methode nach der Signale die Spannung kodieren unterscheiden sich substantiell von der Methode mit DAC. Die PWM-Methode verwendet ein Taktsignal und ein Richtungssignal. Das Taktsignal liefert den absoluten Wert für das Sinussignal und das Richtungssignal liefert den Wert ob die Richtung positiv oder negativ sein soll. Das folgende Diagramm zeigt das Takt- und das Richtungssignal für eine einzelne Phase.



In diesem Diagramm wurde das PWM Taktsignal gefiltert um es von einem digitalen variablen Arbeitszykluswellensignal zu einem analogen Signal zu konvertieren.

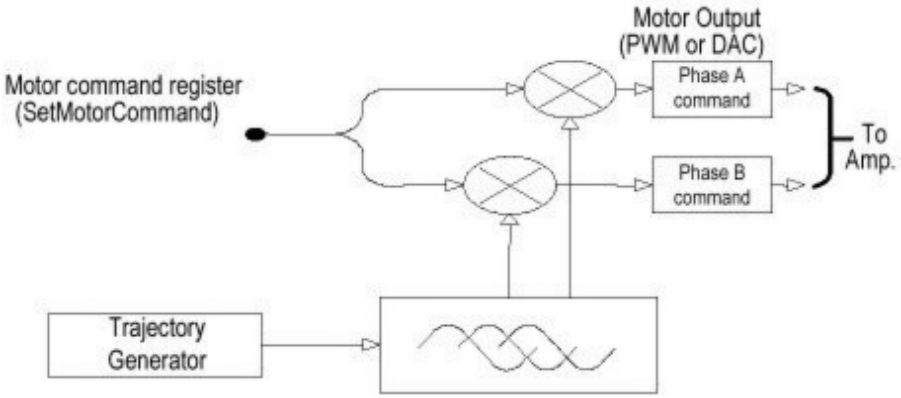
Vor dem Filtern enthält dieses Signal eine impuls-breiten kodierte Darstellung der gewünschten analogen Spannung. In dieser Kodierng bestimmt der Arbeitszyklus der Sinuskurve die erwünschte Spannung. Der PWM-Zyklus hat eine Frequenz von 20 kHz mit einer Auflösung von 10Bit oder 1/1024, oder 78 kHz mit einer Auflösung von 8 Bit oder 1/256.

¤

Mikroschritt Sinus/Cosinus Motorausgangssignale

Mikroschritt Sinus/Cosinus Motorausgangssignale

Zusätzlich zur Trajektoriegeneration unterstützen die POSYS® im Mikroschrittmodus die direkte interne Generierung von Signalen für den Mikroschrittbetrieb für 2- als auch 3-Phasen Schrittmotoren mit Sinus/Cosinus Signalen.
Das folgende Diagramm zeigt einen Überblick über das Mikroschrittschema:



Der Mikroschrittteil des Reglers generiert ein sinusförmiges Signal mit einer Anzahl charakteristischer Ausgangswerte pro Vollschritt (ein Vollschritt ist ein Viertel einer elektrischen Periode). Die Anzahl von Mikroschritten pro Vollschritt wird mit dem Kommando SetPhaseCounts definiert. Die Parameter für dieses Kommando repräsentieren die Anzahl Mikroschritte pro elektrischer Periode (4 Mal die erwünschte Anzahl von Mikroschritten). So sollen z. B. 64 Mikroschritte pro Vollschritt eingestellt werden, dann muss der Befehl SetPhaseCounts 256 benutzt werden. Die maximale Anzahl von Mikroschritten, die pro Vollschritt generiert werden können, ist 256. Der entsprechende Parameter lautet dann 1024.

Die Ausgangsfrequenzen des Mikrochrittsignals werden vom Trajektoriegenerator kontrolliert. Die Amplitude des Mikroschrittsignals wird kontrolliert anhand eines Registers der vom Host gesetzt werden kann (Motor Command Register). Nachstellung dieses Registers vom Host erlaubt unterschiedliche Motorleistungspegel während z.B. der Bewegung und des Stillstands.

Zwei Mikroschrittausgangssignale werden unterstützt, eines für traditionell 2-Phasen Schrittmotoren mit 90° Phasenverschiebung und eines für 3-Phasen Schrittmotoren und AC Induktionsmotoren mit 120° Phasenverschiebung. Für zusätzliche Informationen bzgl. AC Induktionsmotoren verweisen wir auf das Kapitel AC Induction Motor Control.

¤

Mikroschrittausgangssignale

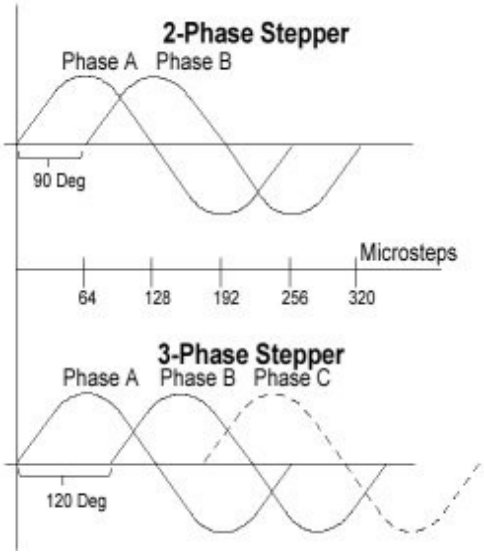
Mikroschrittausgangssignale

Das Kommando SetNumberPhases 2 stellt den Controller auf die Verwendung mit 2-Phasen Schrittmotoren ein, für 3-Phasen Schrittmotoren heisst das Kommando SetNumberPhases 3. Diese Kommando wird nur für die POSYS® 8xx-B und 9xx Multiple Motor, Mikroschritt und bürstenlose Servoversionen benötigt.

Für die POSYS® 182x/192x wird nur noch das Kommando SetMotorType verwendet um automatisch die Anzahl Phasen zu bestimmen.

Phasenanzahl	Motorausgabemodus	Anzahl Ausgangssignale und Name
2-Phasen	PWMSign/Mag	2 (A, B)
2-Phasen	DAC	2 (A, B)
3-Phasen	PWM50/50	3 (A, B, C)
3-Phasen	DAC	2 (A, B)

Die folgende Grafik zeigt die Phasen A und B für ein 2-Phasen Schrittmotor und die Phasen A, B und C Signale für ein 3-Phasen Schrittmotor oder AC Induktionsmotor.



Die dritte Phase C für 3-Phasen Schritt- und AC Induktionsmotoren muss extern durch die Formel C = -(A+B) errechnet werden. Gewöhnlich wird dies vom Treiber übernommen. Im Kapitel "Motor Output" erhalten Sie hierzu mehr Informationen.

¤

Motorsignalausgangskontrolle

Motorsignalausgangskontrolle

Die POSYS® Mikroschrittcontroller unterstützen die Funktion das Niveau der Motorausgabe zu regulieren. Dies ist oft dann nützlich um das Drehmoment, den Verbrauch, die Hitzeentwicklung des Motors im Stillstand oder verschiedene Bewegungszustände zu optimieren.

Der Motorausgangspegel wird vom Motorbefehlsregister kontrolliert. Dieses Register kann mit dem Kommando SetMotorCommand gesetzt werden. Ein Wert zwischen 0 und 32767 repräsentiert eine Verstärkung zwischen 0% und 100%. Das Kommando SetMotorCommand ist doppelt gebuffert und benötigt daher zusätzlich ein Update-Kommando oder einen Breakpoint. Dieses Merkmal kann vorteilhaft sein, wenn es gewünscht wird, dass der Motorstrom sich ändert, dass

man mit anderen Profiländerungen synchronisiert wie am Anfang oder dem Ende einer Bewegung.

Das Ändern des Leistungspegels beeinflusst nicht die Phasen des Mikroschrittausgangssignals oder die Frequenz, es justiert nur die Magnitude des Signals.

¤

AC Induction Motor Control

AC Induction Motor Control

Die POSYS® Mikroschrittcontroller können als variabler Geschwindigkeits-3-Phasen- AC-Asynchronmotor-Regler verwendet werden. In diesem Modus wird der Regler für die Verwendung mit 3 Phasen konfiguriert und spricht den Motor an als wäre es ein Schrittmotor. Die Position des Motors kann nicht sonderlich genau eingehalten werden, aber die Geschwindigkeit lässt sich mit einer Genauigkeit von innerhalb 10%-20% kontrollieren. Ein solcher Controller kann für Spindel oder andere Applikationen wo Geschwindigkeit, nicht Positionsgenauigkeit verlangt wird, eingesetzt werden.

Wenn ein AC-Asynchronmotor mit variable Geschwindigkeitskontrolle verwendet wird, sollte darauf geachtet werden, dass das Motorausgangssignal nie eine Frequenz gleich 0 hat. Sogar wenn der Motor steht, sollte eine gewisse Rotationsfrequenz vorhanden sein. Dies ist weil eine relative Differenz in der Frequenz der Treibersignale und des Rotors (Slip Frequency) erforderlich ist, um eine magnetische Feldsättigung im Stillstand zu vermeiden, welches eine potentielle Gefahr für einen Motorschaden darstellt.

Die POSYS® Mikroschrittcontroller unterstützen nicht "Flux Vector Control" von AC-Asynchronmotoren, nur variable Geschwindigkeitskontrolle. Daher sollte die POSYS® nicht in Verbindung mit AC-Asynchronmotoren verwendet werden wenn präzises Positionieren erforderlich ist.

¤

Command Summary

Command Summary

Folgende Kommandos finden Anwendung im Mikroschrittbetrieb:

Kommando	Funktion
Set/GetOutputMode	Setzt den Ausgang auf entweder analog oder impuls-breiten-moduliert
Set/GetNumberPhases Set/GetMotorType	Setzt die Anzahl der Motorphasen
Set/GetMotorCommand	Setzt die Amplitude des Ausgangssignals. Dieses Kommando ist gebuff und benötigt ein Update
Set/GetPhaseCounts	Setzt die Anzahl Mikroschritte pro elektrischer Periode.

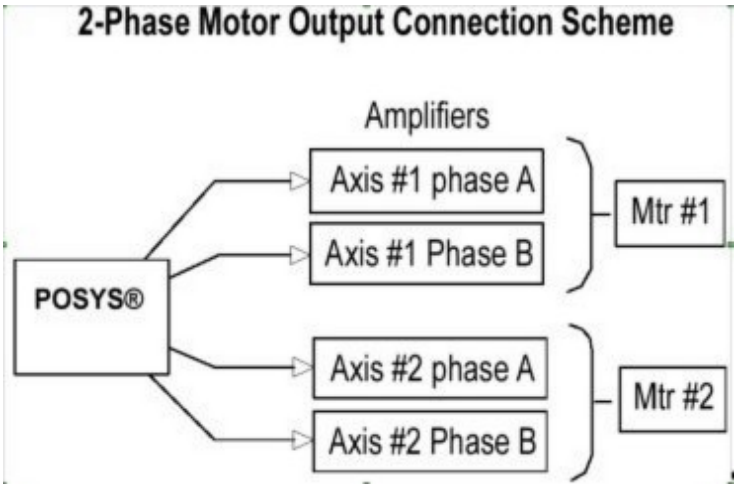
Ehe der Controller irgendetwas ausgibt müssen der Ausgabepegel, der Ausgabemodus und die Anzahl Phasen gesetzt sein.

¤

Motor Treiber Konfigurationen

Motor Treiber Konfigurationen

Nachfolgend ist eine typische Verstärkerkonfiguration für ein 2-Phasen Schrittmotor für entweder PWM oder DAC.



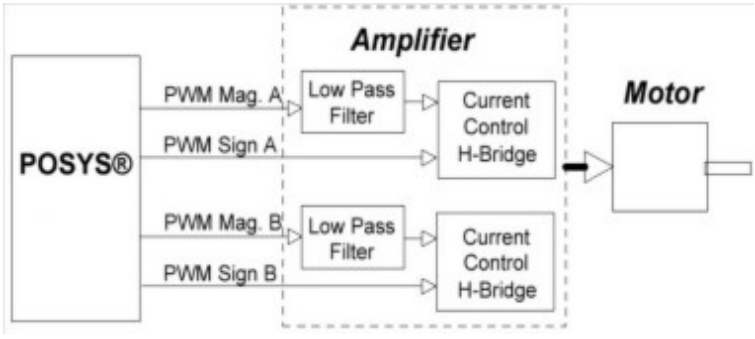
Bei Verwendung eines DACs wird der digitale Motorausgabepegel für jede Phase gewöhnlich konvertiert in ein Analogsignal mit einem Wert zwischen -10V und +10V. Dieses Signal kann dann auf einen handelsüblichen DC-Servoverstärker ausgegeben werden (ein Verstärker für jede Phase) oder in irgendeinen linearen oder schaltenden Verstärker welches Drehmomentkontrolle ausübt und bipolare Ausgabe unterstützt.

In diesem Schema treibt ein Verstärker eine Phase des Schrittmotors an und der Controller generiert die erforderlichen sinusförmigen Signale für jede Phase um eine saubere und akkurate Bewegung zu gewährleisten.

Falls der PWM-Ausgabemodus verwendet wird, werden das Takt- und Richtungssignal für gewöhnlich mit einem H-Bridge-typischen Gerät verbunden. Für maximale Leistung sollte der Verstärker auf Drehmomentsollwert eingestellt sein. Dies minimiert die Stromverzerrung der Spule aufgrund von Induktivität und EMF. Obwohl es einige Methoden gibt die benutzt werden können, um Drehmomentsollwertskontrolle mit PWM zu erreichen, ist es eine übliche Methode das PWM-Signal durch einen Low Pass Filter zu schicken, dabei ein analoges Referenzsignal erstellend welches direkt mit dem Strom durch die Spule verglichen werden kann.

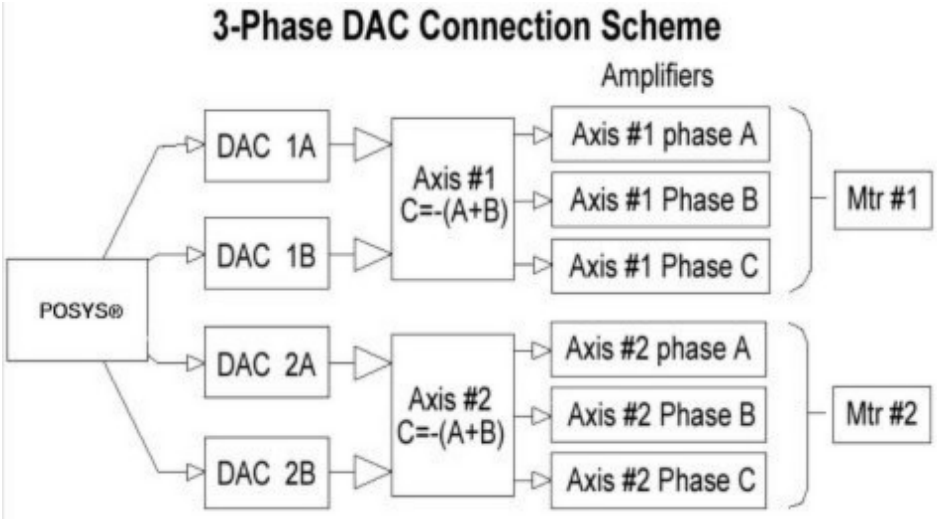
Diverse Verstärker sind verfügbar die kompatibel sind mit diesen Eingangssignalen. Diese Verstärker benötigen einen analogen Referenzeingang (low-passed PWMMag Signal vom Controller) als auch ein Bit mit Vorzeichen (PWMSign vom Controller). Der Verstärker generiert intern einen Drehmomentsollwert anhand eines PWM Drive Schemas.

Das nachfolgende Diagramm zeigt ein solches Drive Schema:



Im Vergleich zur DAC-Methode hat die PWM-Variante in Verbindung mit diesem Verstärkerschema den Vorteil hohe Leistung zu erbringen bei einem Minimum an externen Teilen.

Nachfolgend ist eine typische Verstärkerkonfiguration mit den Mikroschrittversionen der POSYS® in Verbindung mit einem 3-Phasen Schrittmotor oder für einen AC-Asynchronmotor mit 3 Phasen.



Bei der DAC-Methode wird im Controller der digitale Wert mit Hilfe der DACs in einen analogen Wert konvertiert. Für jede Achse sind 2 DAC-Kanäle erforderlich. Die dritte Phase wird mit der Formel C = -(A+B) extern generiert. Dies wird mit Hilfe eines Operationsverstärkerschaltkreises On-board bewerkstelligt.

Für zusätzliche Information oder Hilfe kontaktieren Sie bitte POSYS Motion Control GmbH & Co.KG (www.halbeck.com) unter +49 9158 1612. Sie können Ihre Anfrage auch emailen: support@halbeck.com.

¤

Programmierhandbuch

Wie dieses Handbuch verwendet wird

Instruktionshinweis

Wie benutzt man dieses Handbuch

Dieses Dokument ist in zwei Teile geteilt: zuerst eine detaillierte Beschreibung aller Kommandos aufgeteilt nach Ihrer Funktion und zweitens, eine tabellarische Auflistung.

Im Nachschlageteil sind die Kommandos alphabetisch in ihren entsprechenden Kategorien geordnet, mit der Ausnahme, dass alle "Set/Get"-Kommandopaare (z.B. SetVelocity und GetVelocity) zusammen beschrieben werden. Jede Beschreibung beginnt auf einer neuen Seite und die meisten Kommandos benötigen auch nicht mehr als eine Seite für die Beschreibung. Die Beschreibung ist folgendermaßen gegliedert:

Name	Die Anweisungen sind links angeordnet
Syntax	Die Anweisungen werden mit ihren dazugehörigen Argumenten gelistet
Motortyp	Die Motortypen mit denen das Kommando verwendet werden kann.
Argumente	Es gibt zwei Arten von Argumenten: encoded-field und numerisch Encoded-field Parameter werden in ein einzelnes 16-Bit Daten-Wort gepackt, mit Ausnahme für Ach welches die Bits 11-8 des Instruktionswortes belegt. Der Name des Parameters ist je welcher der in der generellen Syntax angezeigt wird der verwendet wird den Wert de Daten zu repräsentieren. Encoding ist der dem Feld für jenen Fall zugewiesene Wert. numerische Argumente, der Parameterwert, der Typ (Integerwert mit oder ohne Vorzeichen) und Bereich (Range) für akzeptierbare Werte sind gegeben. Numerische Argumente können ein oder zwei Datenwörter benötigen. Für 32-Bit Argumente werd die höherrangigen Anteile zuerst gesandt.
Buffered	Bestimmte Parameter und andere Daten sind gebuffert. Diese Kommandos werden ni eher ausgführt bis ein Update oder MultiUpdate Kommando ausgeführt wird.
Definition	Bezeichnet was die Anweisung ausführt und beschreibt besondere Informationen bezüglich der Anweisung.
Restriktion	Bezeichnet die Umstände, in denen die Anweisung nicht zulässig ist, das bedeutet wa es nicht verwendet werden soll. z.B. Geschwindigkeit, positive/negative Beschleunigu und Jerk-Kommandos dürfen nicht verwendet werden, während ein S-Kurven Profil ausgeführt wird und die Achse in Bewegung ist.
MotionScript® Aufrufkonvention und DLL Aufrufkonvention	Die Syntax für das Kommando in Delphi, C ++, VB in der POSYSxxx.DLL um das Kommando aus Ihrem Programm heraus ausführen zu können. Auch die Syntax für die Einbindung in mit MotionScript® erstellte Programme wird dargestellt.
siehe	Verweist auf verwandte Kommandos

¤

Kommandos

Liste der verfügbaren Kommandos

POSYS® Functions and Procedures

MotionScript® requires for the parameters the type DOUBLE and if a value is returned it is also returned as a DOUBLE. By default all numeric MotionScript® variables are of the type DOUBLE, in fact that is the only numeric type

Card Initialization & Addressing

Procedure InitializePOSYS(cardAddress0, cardAddress1:word);stdcall;
Function OpenDevice(base_IO:word;base_MEM, MEM_range:DWORD):P1800_HANDLE;stdcall; //only valid for POSYS 1800 equipped with Dual Port RAM
Function OpenDevice:word;
Function CloseDevice(hPosys :integer) :integer;stdcall;
Function DetectPosysDevices:word;stdcall;
Function RegisteredPosysDevices:word;stdcall;

Breakpoints & Interrupts

Procedure ClearInterrupt(hPosys:Thandle);stdcall;
Function GetInterruptAxis(hPosys:Thandle):word;stdcall;
Procedure SetBreakpoint(hPosys:Thandle;axis,breakpoint,sourceAxis,action,trigger:word);stdcall;
Function GetBreakpoint(hPosys:Thandle;axis,breakpoint:word):word;stdcall;
Procedure SetBreakpointValue(hPosys:Thandle;axis,breakpoint:word;value:integer);stdcall;
Function GetBreakpointValue(hPosys:Thandle;axis,breakpoint:word):integer;stdcall;
Procedure SetBreakpointUpdateMask(hPosys:Thandle;axis,breakpointID,mask:word);stdcall;
Function GetBreakpointUpdateMask(hPosys;Thandle;axis,breakpointID:word):word;stdcall;
Procedure SetInterruptMask(hPosys:Thandle;axis,interruptMask:word);stdcall;
Function GetInterruptMask(hPosys:Thandle;axis:word):word;stdcall;

Commutation

Function GetPhaseCommand(hPosys:Thandle;axis:word):integer;stdcall;
Procedure InitializePhase(hPosys:Thandle;axis:word);stdcall;
Procedure SetCommutationMode(hPosys:Thandle;axis,mode:word);stdcall;
Function GetCommutationMode(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetNumberPhases(hPosys:Thandle;axis,phases:word);stdcall;
Function GetNumberPhases(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetPhaseAngle(hPosys:Thandle;axis,angle:word);stdcall;
Function GetPhaseAngle(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetPhaseCorrectionMode(hPosys:Thandle;axis,mode:word);stdcall;
Function GetPhaseCorrectionMode(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetPhaseCounts(hPosys:Thandle;axis,count:word);stdcall;
Function GetPhaseCounts(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetPhaseInitializeMode(hPosys:Thandle;axis,mode:word);stdcall;
Function GetPhaseInitializeMode(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetPhaseInitializeTime(hPosys:Thandle;axis,time:word);stdcall;
Function GetPhaseInitializeTime(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetPhaseOffset(hPosys:Thandle;axis,offset:word);stdcall;
Function GetPhaseOffset(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetPhasePrescale(hPosys:Thandle;axis,mode:word);stdcall;
Function GetPhasePrescale(hPosys:Thandle;axis:word):word;stdcall;

Digital Servo Filter

Procedure ClearPositionError(hPosys:Thandle;axis:word);stdcall;
Procedure DACOutputsOff(hPosys:Thandle);stdcall;
Procedure DACOutputsOn(hPosys:Thandle);stdcall;
Function GetAmpEnable(hPosys:Thandle;ampID:word):word;stdcall;
Function GetDerivative(hPosys:Thandle;axis:word):word;stdcall;
Function GetIntegral(hPosys:Thandle;axis:word):integer;stdcall;
Function GetPositionError(hPosys:Thandle;axis:word):integer;stdcall;
Procedure SetAmpDisable(hPosys :integer; disable :word);stdcall;
Procedure SetAmpEnable(hPosys :integer; enable :word);stdcall;
Procedure SetAutoStopMode(hPosys:Thandle;axis,mode:word);stdcall;
Function GetAutoStopMode(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetBiQuadCoefficient(hPosys:Thandle;axis,coefficientID,filterID,value:word);stdcall;
Function GetBiQuadCoefficient(hPosys:Thandle,axis:word);stdcall;
Procedure SetDerivativeTime(hPosys:Thandle;axis,time:word);stdcall;
Function GetDerivativeTime(hPosys:Thandle;axis:word):word;stdcall;

```
Procedure SetIntegrationLimit(hPosys:Thandle;axis:word;limit:integer);stdcall;
Function GetIntegrationLimit(hPosys:Thandle;axis:word):integer;stdcall;
Procedure SetKaff(hPosys:Thandle;axis,Kaff:word);stdcall;
Function GetKaff(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetKd(hPosys:Thandle;axis,Kd:word);stdcall;
Function GetKd(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetKi(hPosys:Thandle;axis,Ki:word);stdcall;
Function GetKi(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetKp(hPosys:Thandle;axis,Kp:word);stdcall;
Function GetKp(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetKout(hPosys:Thandle;axis,Kout:word);stdcall;
Function GetKout(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetKvff(hPosys:Thandle;axis,Kvff:word);stdcall;
Function GetKvff(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetMotorBias(hPosys:Thandle;axis,bias:word);stdcall;
Function GetMotorBias(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetMotorLimit(hPosys:Thandle;axis,limit:word);stdcall;
Function GetMotorLimit(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetPositionErrorLimit(hPosys:Thandle;axis:word;limit:integer);stdcall;
Function GetPositionErrorLimit(hPosys:Thandle;axis:word):integer;stdcall;
```

Piezo (Ceramic) Motor Functions

```
Procedure SetpzOffsetPos(hPosys:Thandle;axis:word;offset:word);stdcall;external DLL;
Function GetpzOffsetPos(hPosys:Thandle;axis:word):word;stdcall;external DLL;
Procedure SetpzOffsetNeg(hPosys:Thandle;axis:word;offset:word);stdcall;external DLL;
Function GetpzOffsetNeg(hPosys:Thandle;axis:word):word;stdcall;external DLL;
Procedure SetpzIntLimitStl(hPosys:Thandle;axis:word;limit:longword);stdcall;external DLL;
Function GetpzIntLimitStl(hPosys:Thandle;axis:word):longword;stdcall;external DLL;
Procedure SetpzWindow(hPosys:Thandle;axis:word;window:word);stdcall;external DLL;
Function GetpzWindow(hPosys:Thandle;axis:word):word;stdcall;external DLL;
Procedure SetpzKp(hPosys:Thandle;axis:word;kp:word);stdcall;external DLL;
Function GetpzKp(hPosys:Thandle;axis:word):word;stdcall;external DLL;
Procedure SetpzKi(hPosys:Thandle;axis:word;ki:word);stdcall;external DLL;
Function GetpzKi(hPosys:Thandle;axis:word):word;stdcall;external DLL;
Procedure SetpzKd(hPosys:Thandle;axis:word;kd:word);stdcall;external DLL;
Function GetpzKd(hPosys:Thandle;axis:word):word;stdcall;external DLL;
Procedure SetpzKvff(hPosys:Thandle;axis:word;kvff:word);stdcall;external DLL;
Function GetpzKvff(hPosys:Thandle;axis:word):word;stdcall;external DLL;
Procedure SetpzKaff(hPosys:Thandle;axis:word;kaff:word);stdcall;external DLL;
Function GetpzKaff(hPosys:Thandle;axis:word):word;stdcall;external DLL;
Procedure SetpzIntLim(hPosys:Thandle;axis:word;intlim:longword);stdcall;external DLL;
Function GetpzIntLim(hPosys:Thandle;axis:word):longword;stdcall;external DLL;
Procedure SetpzKdPeriod(hPosys:Thandle;axis:word;kdperiod:word);stdcall;external DLL;
Function GetpzKdPeriod(hPosys:Thandle;axis:word):word;stdcall;external DLL;
Procedure SetpzWindow2(hPosys:Thandle;axis:word;window2:word);stdcall;external DLL;
Function GetpzWindow2(hPosys:Thandle;axis:word):word;stdcall;external DLL;
Procedure SetpzKvff2(hPosys:Thandle;axis:word;kvff2:word);stdcall;external DLL;
Function GetpzKvff2(hPosys:Thandle;axis:word):word;stdcall;external DLL;
```

Encoder

```
Procedure AdjustActualPosition(hPosys:Thandle;axis:word;position:integer);stdcall;
Function GetAbsoluteSSIPosition(hPosys:Thandle;axisID:word):integer;stdcall;
Function GetActualVelocity(hPosys:Thandle;axis:word):integer;stdcall;
Function GetCaptureValue(hPosys:Thandle;axis:word):integer;stdcall;
Function GetSSIVersion(hPosys:Thandle):word;stdcall;
Procedure WriteSSIRegister(hPosys:Thandle;axisID:word; resolution, frequency:word);stdcall;
Procedure ReadSSIRegister(hPosys:Thandle;axisID:word;var resolution:word;var frequency:word);stdcall;
Procedure WriteSSIResolution(hPosys:Thandle;axisID:word;var resolution:word);stdcall;
Procedure ReadSSIResolution(hPosys:Thandle;axisID:word;var resolution:word);stdcall;
Procedure WriteSSIFrequency(hPosys:Thandle;axisID:word;var resolution:word);stdcall;
Procedure ReadSSIFrequency(hPosys:Thandle;axisID:word;var resolution:word);stdcall;
Procedure ResetSSI(hPosys:Thandle);stdcall;
Procedure SetActualToAbsolutePosition(hPosys:Thandle;axisID:word);stdcall;
Procedure SetActualPosition(hPosys:Thandle;axis:word;pos:integer);stdcall;
Function GetActualPosition(hPosys:Thandle;axis:word):integer;stdcall;
Procedure SetActualPositionUnits(hPosys:Thandle;axis,mode:word);stdcall;
Function GetActualPositionUnits(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetAuxiliaryEncoderSource(hPosys:Thandle;axis,mode,auxiliaryAxis:word);stdcall;
Function GetAuxiliaryEncoderSource(hPosys:Thandle;axis:word);stdcall;
Procedure SetCaptureSource(hPosys:Thandle;axis,source:word);stdcall;
Function GetCaptureSource(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetEncoderModulus(hPosys:Thandle;axis,modulus:word);stdcall;
```

```
Function GetEncoderModulus(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetEncoderSource(hPosys:Thandle;axis,source:word);stdcall;
Function GetEncoderSource(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetEncoderToStepRatio(hPosys:Thandle;axis,encodercounts,steps:word);stdcall;
Function GetEncoderToStepRatio(hPosys:Thandle;axis:word):integer;stdcall;
```

On-Board RAM

```
Procedure EnableDPRam(hPosys:P1800_Handle);stdcall;  //only valid for POSYS® 1800 series equipped with Dual Port RAM
Procedure DisableDPRam(hPosys:P1800_Handle);stdcall;  //only valid for POSYS® 1800 series equipped with Dual Port RAM
Function ReadBuffer(hPosys:Thandle;bufferID:word):integer;stdcall;
Procedure SetBufferFunction(hPosys:Thandle;axis:word;funct:integer;bufferID:word);stdcall;
Function GetBufferFunction(hPosys:Thandle;axis:word;funct:word):word;stdcall;
Procedure SetBufferLength(hPosys:Thandle;bufferID:word;length:integer);stdcall;
Function GetBufferLength(hPosys:Thandle;bufferID:word):integer;stdcall;
Procedure SetBufferReadIndex(hPosys:Thandle;bufferID:word;index:integer);stdcall;
Function GetBufferReadIndex(hPosys:Thandle;bufferID:word):integer;stdcall;
Procedure SetBufferStart(hPosys:Thandle;bufferID:word;address:integer);stdcall;
Function GetBufferStart(hPosys:Thandle;bufferID:word):integer;stdcall;
Procedure SetBufferWriteIndex(hPosys:Thandle;bufferID:word;index:integer);stdcall;
Function GetBufferWriteIndex(hPosys:Thandle;bufferID:word):integer;stdcall;
Procedure WriteBuffer(hPosys:Thandle;bufferID:word;value:integer);stdcall;
Function ReadDPRamLong(hPosys:Thandle;offset:dword):integer;stdcall;
Procedure WriteDPRamLong(hPosys:Thandle;offset:dword;value:dword);stdcall;
```

Motor Output

```
Function GetCurrentMotorCommand(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetMotorCommand(hPosys:Thandle;axis,value:word);stdcall;
Function GetMotorCommand(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetMotorMode(hPosys:Thandle;axis,mode:word);stdcall;
Function GetMotorMode(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetMotorType(hPosys:Thandle;axis:word):word;stdcall;
Function GetMotorType(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetOutputMode(hPosys:Thandle;axis,mode:word);stdcall;
Function GetOutputMode(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetPWMFrequency(hPosys:Thandle;axis,frequency:word);stdcall;
Function GetPWMFrequency(hPosys:Thandle;axis:word);stdcall;
Procedure SetStepRange(hPosys:Thandle;axis,frequency:word);stdcall;
Function GetStepRange(hPosys:Thandle;axis:word):word;stdcall;
```

Profile Generation

```
Function GetCommandedAcceleration(hPosys:Thandle;axis:word):integer;stdcall;
Function GetCommandedPosition(hPosys:Thandle;axis:word):integer;stdcall;
Function GetCommandedVelocity(hPosys:Thandle;axis:word):integer;stdcall;
Procedure MultiUpdate(hPosys:Thandle;mask:word);stdcall;
Procedure SetAcceleration(hPosys:Thandle;axis:word;accel:integer);stdcall;
Function GetAcceleration(hPosys:Thandle;axis:word):integer;stdcall;
Procedure SetDeceleration(hPosys:Thandle;axis:word;deceleration:integer);stdcall;
Function GetDeceleration(hPosys:Thandle;axis:word):integer;stdcall;
Procedure SetGearMaster(hPosys:Thandle;axis,masterAxis,source:word);stdcall;
Function GetGearMaster(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetGearRatio(hPosys:Thandle;slaveAxis:word;ratio:integer);stdcall;
Function GetGearRatio(hPosys:Thandle;slaveAxis:word):integer;stdcall;
Procedure SetJerk(hPosys:Thandle;axis:word;jerk:integer);stdcall;
Function GetJerk(hPosys:Thandle;axis:word):integer;stdcall;
Procedure SetPosition(hPosys:Thandle;axis:word;position:integer);stdcall;
Function GetPosition(hPosys:Thandle;axis:word):integer;stdcall;
Procedure SetProfileMode(hPosys:Thandle;axis,mode:word);stdcall;
Function GetProfileMode(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetStartVelocity(hPosys:Thandle;axis:word;velocity:integer);stdcall;
Function GetStartVelocity(hPosys:Thandle;axis:word):integer;stdcall;
Procedure SetStop(hPosys:Thandle;axis,mode:word);stdcall;
Function GetStop(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetStopMode(hPosys:Thandle;axis,mode:word);stdcall;
Function GetStopMode(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetVelocity(hPosys:Thandle;axis:word;velocity:integer);stdcall;
Function GetVelocity(hPosys:Thandle;axis:word):integer;stdcall;
Procedure Up_date(hPosys:Thandle;axis:word);stdcall;
```

```
Procedure Update(hPosys:Thandle;axis:word);stdcall;
```

Servo Loop Control

```
Function GetTime(hPosys:Thandle):integer;stdcall;
Procedure SetAxisMode(hPosys:Thandle;axis,mode:word);stdcall;
Function GetAxisMode(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetLimitSwitchMode(hPosys:Thandle;axis,mode:word);stdcall;
Function GetLimitSwitchMode(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetMotionCompleteMode(hPosys:Thandle;axis,mode:word);stdcall;
Function GetMotionCompleteMode(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetSampleTime(hPosys:Thandle;time:word);stdcall;
Function GetSampleTime(hPosys:Thandle):word;stdcall;
Procedure SetSettleTime(hPosys:Thandle;axis,time:word);stdcall;
Function GetSettleTime(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetSettleWindow(hPosys:Thandle;axis,window:word);stdcall;
Function GetSettleWindow(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetTrackingWindow(hPosys:Thandle;axis,window:word);stdcall;
Function GetTrackingWindow(hPosys:Thandle;axis:word):word;stdcall;
```

Status Registers and AxisOut Indicator

```
Function GetActivityStatus(hPosys:Thandle;axis:word):word;stdcall;
Function GetAxis_In_Out_Home(hPosys:Thandle;statID:word):word;stdcall;
Function GetEventStatus(hPosys:Thandle;axis:word):word;stdcall;
Function GetSignalStatus(hPosys:Thandle;axis:word):word;stdcall;
Procedure ResetEventStatus(hPosys:Thandle;axis,mask:word);stdcall;
Procedure SetAxisOutSource(hPosys:Thandle;axis:word;sourceAxis,bit,reg:word);stdcall;
Function GetAxisOutSource(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetSignalSense(hPosys:Thandle;axis,mask:word);stdcall;
Function GetSignalSense(hPosys:Thandle;axis:word):word;stdcall;
```

Traces

```
Function GetTraceCount(hPosys:Thandle):integer;stdcall;
Function GetTraceStatus(hPosys:Thandle):word;stdcall;
Procedure SetTraceMode(hPosys:Thandle;mode:word);stdcall;
Function GetTraceMode(hPosys:Thandle):word;stdcall;
Procedure SetTracePeriod(hPosys:Thandle;period:word);stdcall;
Function GetTracePeriod(hPosys:Thandle):word;stdcall;
Procedure SetTraceStart(hPosys:Thandle;triggerAxis,condition,triggerBit,triggerState:word);stdcall;
Function GetTraceStart(hPosys:Thandle):word;stdcall;
Procedure SetTraceStop(hPosys:Thandle;triggerAxis,condition,triggerBit,triggerState:word);stdcall;
Function GetTraceStop(hPosys:Thandle):word;stdcall;
Procedure SetTraceVariable(hPosys:Thandle;variableNumber,traceAxis,variable:word);stdcall;
Function GetTraceVariable(hPosys:Thandle;variableNumber:word):word;stdcall;
```

Miscellaneous (Controller Reset, Communication Mode aso)

```
# Function EnableIO(deviceNo, portCount:word):word (no longer needed with new driver Windriver)
Function GetChecksum(hPosys);stdcall;
Function GetHostIOError(hPosys:Thandle):word;stdcall;
Function GetProcessorStatus(hPosys:Thandle):word;stdcall;
Function Get_Version(hPosys:Thandle):integer;stdcall;
Function GetVersion(hPosys:Thandle):integer;stdcall;
Procedure HardReset(hPosys:Thandle);stdcall;
Procedure MCReset(hPosys:Thandle);stdcall;
Procedure NoOperation(hPosys:Thandle);stdcall;
Function PortInW(hPosys:Thandle;address:word):word;stdcall;
Procedure PortOutW(hPosys:Thandle;address,outword:word);stdcall;
Procedure PortOutB(hPosys:Thandle;address:word;value:byte);stdcall;
Function ReadAnalog(hPosys:Thandle;portID:word):word;stdcall;
Function ReadIO(hPosys:Thandle;ioID:word):word;stdcall;
Function ReadPLD(hPosys:Thandle;pld_adr:word):word;stdcall;
Procedure SoftReset(hPosys:Thandle);stdcall;
Procedure SetCANMode(hPosys:mask);stdcall;
Function GetCANMode(hPosys);stdcall;
Procedure SetDiagnosticPortMode(hPosys:Thandle;mode:word);stdcall;
Function GetDiagnosticPortMode(hPosys:Thandle):word;stdcall;
Procedure SetSerialPort(hPosys:Thandle;baudrate,parity,stopbits,protocol,mdaddress:word);stdcall;
Function GetSerialPort(hPosys:Thandle):word;stdcall;
Procedure SetSerialPortMode(hPosys:Thandle;baudrate,parity,stopbits,protocol,mdaddress:word);stdcall;
```

```
Function GetSerialPortMode(hPosys:Thandle):word;stdcall;
Procedure SetSynchronizationMode(hPosys:Thandle;mode:word);stdcall;
Function GetSynchronizationMode(hPosys:Thandle):word;stdcall;
Procedure SetWatchDog(hPosys:Thandle;watchdog_adr:word;watchdog_value:word);stdcall;
Procedure WriteIO(hPosys:Thandle;ioID,data:word);stdcall;
```

Additional MotionScript® Functions

```
Procedure BitReset(hPosys:Thandle;testVar,bitToReset:integer);stdcall;
Procedure BitSet(hPosys:Thandle;testVar,bitToReset:integer);stdcall;
Procedure BitTest(hPosys:Thandle;testVar,bitToReset:integer);stdcall;
Procedure CompareActivity(hPosys:Thandle;axis:word;mask:integer);stdcall;
Procedure CompareEvent(hPosys:Thandle;axis:word;mask:integer);stdcall;
Procedure CompareSignal(hPosys:Thandle;axis:word;mask:integer);stdcall;
```

¤

Karteninitializierung und -adressierung

InitializePOSYS

nur benötigt für ISA-Bus und PC/104 Karten (POSYS® 7xx, 8xx, 8xx-B und 18xx)
nur benötigt mit Kithara-Treiber (nicht mehr im Einsatz)

Syntax	InitializePOSYS(hPosys, cardAddress) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle		
		Type		
	card address	unsigned 16 bits	Range	512-1016
			Scaling	none
Definition	InitializePOSYS initialisiert den Adressbereich, welches die Positioniersteuerkarte im Computer belegt. Die cardAddress muss der Adresse entsprechen die auf der Karte eingestellt wurde. Dies entspricht der Basisadresse der Karte und ein Aufruf von InitializePOSYS(cardAddress) setzt die Kommando-, Daten und HardwareReset Adressen. Keine andere Positioniersteuerkarte oder anderes Peripheriegerät sollte einer dieser Adressen belegen (cardAddress + 7 Bytes). Falls mehr als eine Karte installiert ist, müssen 8 Bytes zwischen der Karte 1 und der Karte 2 liegen. z.B. Karte 1 hat die Adresse 768 (300H), dann sollte die Karte 2 die Adresse 776 (308H) haber			
Einschränkung	Die einzigen Restriktionen beziehen sich auf den zur Verfügung stehenden Adressbereich des eingesetzten Computers. Theoretisch ist es auch möglich Adressbereiche unterhalb 512 zu verwenden, solange kein Konflikt mit anderen im C omputer befindlichen Peripheriegeräte auftauchen. Diese Prozedur ist mit dem neuen WinDriver Treiber nicht mehr von nöten. Es wurde nur beibehalten aus Rückwärtskompatibilitätsgründen für ISA- bzw. PC /104 Karten. Das komman OpenDevice/CloseDevice übernimmt diese Funktion.			
siehe	OpenDevice/CloseDevice			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	InitializePOSYS(hPosys#, cardAddress) hPosys is the value returned by a call to EnableIO			

¤

OpenDevice
CloseDevice

Syntax	OpenDevice(deviceNumber) stdcall			für PCI-Bus
	OpenDevice(base_IO;base_MEM;MEM_range) stdcall			für POSYS® 1800 mit Dual Port RAM (PC-104 bus)
	OpenDevice(base_IO) stdcall			für POSYS® 700/ 800/800-B/1800 (ISA & PC-104 bus; 1800 mit Single Port RAM)
	CloseDevice(hPosys) stdcall			für alle Karten
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	PCI: deviceNumber		keine Begrenzung (1 -)	
	ISA/PC-104:			
	base_IO	unsigned 16 Bit	512 - 1024	
	base_MEM	unsigned 32 Bit	B0000 bis EFFFF	
	MEM_range	unsigned 32 Bit	16k x 16 (32768 KB) oder andere Größen falls Karte damit ausgestattet wurde	
Ergebnis	PCI: Liefert ein 32 Bit Handle zurück, "hPosys", die mit der Kartentreiberinstanz "deviceNumber" geöffnet wurde.			
	ISA/PC-104: reserviert den angegebenen Bereich für die ISA-Bus basierten Karten (ISA & PC/104). Zusätzlich kann bei der POSYS® 1800 die zu verwendende Basisspeicheradresse und der zu verwendende Speicherbereich reserviert werden, sofern es sich um eine mit DPRAM bestückte POSYS® 1800 handelt. Mehr hierzu im Kapitel "Verwendung des Dual Port RAM mit der POSYS® 1800"			
Definition	PCI: OpenDeviceöffnet eine Instanz für die POSYS® PC I-Treiber und liefert einen Handle, "hPosys", um die Karte ansprechen zu können.			
	ISA/PC-104: Startet den Treiber. Bei der POSYS® 1800 wird zusätzlich bei mit Dual Port RAM bestückten Karten die Basisspeicheradresse und der Speicherbereich definiert.			
	CloseDevice(hPosys) schliesst den Gerätetreiber für die spezifizierte Karte im System und gibt die Resourcen frei. Es werden keine Werte zurückgeliefert.			
	(hPosys) zeigt das Gerät an dass geöffnet (nur PC I) oder geschlossen (PC I, ISA/PC-104) wer soll.			
Einschränkung	Die Anzahl unterstützte Karten hängen nur von der zur Verfügung stehenden Steckplätze ab. D Treiber kann theoretisch eine unbegrenzte Anzahl Instanzen öffnen.			
	Für die POSYS® 1800 können 2 Methoden des Aufrufes gültig sein, die abhängig von der Art d Speicherbestückung sind. Mit Dual Port RAM bestückte Karten können beide Methoden verwend werden, wobei wenn die Karte mit DPRAM bestückt ist, und der Speicher als solcher nicht reserviert wird, es dann nur als SPRAM angesprochen werden kann.			
	Dagegen darf der Treiber für mit SPRAM bestückte POSYS® 1800 nur mit dem base_IO Parameter geöffnet werden. base_MEM und MEM_range dürfen dann nicht verwendet werden.			
siehe	PortInOutW, DetectPosysDevices, RegisteredPosysDevices			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	OpenDevice(deviceNumber)			für PCI-Bus
	OpenDevice(base_IO;base_MEM;MEM_range)			für POSYS® 1800 mit DPRAM
	OpenDevice(base_IO)			für POSYS® 700/800/800-B/ 1800 mit SPRAM
	CloseDevice(hPosys#)			für alle POSYS®

¤

DetectPosysDevices

nur gültig für POSYS® PCI Karten (POSYS® 900/1900)

Syntax	DetectPosysDevices stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	none			
Ergebnis	Liefert die Anzahl gefundene POSYS® PCI Positioniersteuerungen. Es wurden keine gefunden falls 0 zurückgeliefert wird.			
Definition	DetectPosysDevices sucht nach allen installierten POSYS® PCI Positioniersteuerkarten. Es werden alle PCI Busse abgesucht.			
Einschränkung	Der Treiber unterstützt nun theoretisch eine unbegrenzte Anzahl POSYS® PCI Positioniersteuerkarten und meldet sie als 1 - xxx zurück.			
siehe	OpenDevice/CloseDevice, RegisteredPosysDevices			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	DetectPosysDevices()			//keine Parameter

¤

RegisteredPosysDevices

nur gültig für POSYS® ISA/PC-104 Karten (POSYS® 7xx/8xx/8xx-B/18xx)

Syntax	RegisteredPosysDevices stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	none			
Ergebnis	Liefert eine Zahl zwischen 0 und 16 zurück Meldet die Anzahl der gefundenen POSYS® ISA/PC-104 Positioniersteuerungen. Wenn keine gefunden wurden, dann wird eine 0 zurückgeliefert.			
Definition	RegisteredPosysDevices findet alle registrierten POSYS® ISA/PC -104 Positioniersteuerunger Alle registrierten Karten, die mit POSYSDriverXSetup registriert wurden, werden gefunden.			
Einschränkung	Der Treiber unterstützt theoretisch eine unbegrenzte Anzahl von POSYS® ISA/PC-104 Positioniersteuerkarten aber POSYSDriverXSetup ist auf eine maximale Anzahl von 16 Karten begrenzt und meldet sie als 1 - 16 zurück. Falls mehr Karten eingesetzt werden sollen, kann POSYSDriverXSetup entsprechend angepasst werden.			
siehe	OpenDevice/CloseDevice			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	RegisteredPosysDevices()			

POSYS® Controller (ISA-Bus und PC/104) müssen mit dem Programm POSYSDriverXSetup registriert worden sein.

¤

Breakpoints und Interrupts

ClearInterrupt

ClearInterrupt

Syntax	ClearInterrupt(hPosys) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
Definition	ClearInterruptsetzt den HostInterrupt in den inaktiven Status zurück. Falls weitere Interrupts unbearbeitet sind, wird der HostInterrupt innerhalb eines Zyklus in den aktiven Status zurückkehren. Es wird benutzt nachdem ein Interrupt erkannt und vom Host bearbeitet wurde. Dieses Kommando hat keinen Einfluss auf den Event Status Register. Wenn dieses Kommando ausgeführt wird, ohne dass Interrupts unbearbeitet anstehen, wird das Kommando ignoriert. Bei der Kommunikation via C AN, resettet dieses Kommando den Interrupt Message Sent Flag. Wenn ein Interrupt an einer Achse getriggert wurde, wird eine einzelne Interrupt Message gesendet und keine weitere Aktion an der Achse ausgeführt bis das Kommando bearbeitet wur Im Kommunikationsmodus Parallel (PC -Bus) oder Seriell wird der Parameter für Axis nicht benötigt.			
Einschränkung				
siehe	GetInterruptAxis, Set/GetInterruptMask, ResetEventStatus			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	ClearInterrupt(hPosys#)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

GetInterruptAxis

Syntax	GetInterruptAxis(hPosys) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
Ergebnis	Name	Instance	Encoding	
	axis	None Axis1mask Axis2mask Axis3mask Axis4mask	0 1 2 4 8	
Definition	GetInterruptAxisliefert einen Wert zurück der alle Achsen mit unbearbeiteten Interrupts anzeigt. Die Achsennummern sind den niederwertigen 4 Bits des zurückgelieferten Datenworte zugewiesen. Bits die einer unterbrechenden Achse zugewiesen werden können, sind mit einer belegt. Falls kein Interrupt ausgelöst wurde, ist der zurückgelieferte Wert = 0.			
Einschränkung				
siehe	ClearInterrupt, Set/GetInterruptMask			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	GetInterruptAxis(hPosys#)			

¤

Set/GetBreakpoint

SetBreakpoint
GetBreakpoint

manche Optionen sind Firmware-abhängig für POSYS® 1800/1900 falls mit Motion Control Chipsatz bestückt >< V2.x

---

Syntax	SetBreakpoint(hPosys, axis, breakpoint, sourceAxis, action, trigger) stdcall GetBreakpoint(hPosys, axis, breakpoint) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1		0
		Axis2		1
		Axis3		2
		Axis4		3
	breakpointID	Breakpoint 1		0
		Breakpoint 2		1
	sourceAxis	Axis1		0
		Axis2		1
		Axis3		2
		Axis4		3
	action	(none)		0
		Update		1
		AbruptStop		2
		SmoothStop		3
		Motor Off **		4
		Disable Position Loop & Higher Modules *		5
		n/a		6
		Disable Motior Output & Higher Modules *		7
		Abrupt Stop with Position Error Clear *		8
	trigger	(none)		0
		GreaterOrEqualCommandedPosition		1
		LesserOrEqualCommandedPosition		2
		GreaterOrEqualActualPosition		3
		LesserOrEqualActualPosition		4
		CommandedPositionCrossed		5
		ActualPositionCrossed		6
		Time		7
		EventStatus		8
		ActivityStatus		9
		SignalStatus		Ah
		Drive Status		Bh

Definition

SetBreakPointveranlasst einen Breakpoint, aufgrund einer Bedingung an der Quellachse, für spezifizierte Achse getriggert zu werden, welches die selbe Achse oder eine andere Achse sein kann. Bis zu 2 Breakpoints können pro Achse gleichzeitig gesetzt werden.

Die 6 Breakpoints für Position und der Time Breakpoint sind Schwellwert getriggert; die Breakpoints werden ausgelöst, wenn ein bestimmter Wert erreicht, oder überschritten wird. Di Status Breakpoints sind Level getriggert; der Breakpoint wird ausgelöst wenn ein bestimmter E oder eine Bit-Kombination seinen Status im Status Register den Zustand wechselt. Beides, Schwellwert und Bitspezifikationen werden mit dem Kommando SetBreakPointValue gesetzt.

Action bestimmt was die POSYS® unternehmen soll wenn ein Breakpoint ausgelöst wird:

Action	resultierende Kommandosequenz
None	keine Aktion
Update	Update axis
AbruptStop	SetStop axis, AbruptStop, Update axis
SmoothStop	SetStop axis, SmoothStop, Update axis
MotorOff **	SetMotorMode axis, Off, Update axis
Disable Position Loop & Higher Modules *	Deaktiviert Bahngenerator und höhere Module

Disable Motor Output & Higher Modules *	Deaktiviert alle Module inkl. Motorausgabe
Abrupt Stop with Position Error Clear *	Abrupter Stop des Bahngenerators und Schleppabstandsfehler wird genullt

Axis ist die Achse für welches der Breakpoint definiert wurde.

GetBreakPointliest die Bedingung, Axis und Action für den spezifizierten Breakpoint (1 oder 2 für die angezeigte Achse. Sobald ein Breakpoint aktiviert wird, wird der "trigger value" auf Null zurück gesetzt. Die CommandedPositionCrossed und ActualPositionCrossed Trigger werden zu einem der positionsabhängigen Triggerversionen 1-4 konvertiert, je nachdem welche Position aktuell war beim Ausführen des Kommandos.

Zwei völlig unterschiedliche Breakpoints werden unterstützt, jedes ein anderer Breakpointtyp u mit unterschiedlichem Vergleichswert. Das Breakpointfeld spezifiziert welches Breakpoint die Kommandos SetBreakPoint und GetBreakPoint adressiert.

Einschränkung

Ehe eine neue Breakpointbedingung gesetzt werden kann (Kommando SetBreakPoint), muss grundsätzlich vorher der Vergleichswert geladen werden (Kommando SetBreakPointValue). Dies ist deswegen notwendig, weil sobald die Bedingung für den Breakpoint gesetzt wurde, die POSYS® den BreakPointValue Register verwendet und wenn bis dahin der Wert nicht entsprechend definiert wurde, wird sich der Breakpoint auch nicht wie erwartet verhalten.

* mit Asterisk gekennzeichnete Optionen unter Action sind nur verfügbar in der Firmware Version >2.x für POSYS® 1800/1900.
** mit 2 Asterisk gekennzeichnete Optionen unter Action sind nur in der Version POSYS® 800-B, 900, 1800 und 1900. Dabei beschränkt sich die Verfügbarkeit bei der 1800/1900 auf Firmwareversionen <2.x.

siehe

Set/GetBreakpointValue, Set/GetBreakpointUpdateMask

DLL: Delphi, C++, VB

wir verweisen auf die verschiedenen Importdateien (imp*.*)

MotionScript® Aufruf-konvention

SetBreakpoint(hPosys#, axis, breakpoint, sourceAxis, action, trigger)
GetBreakpoint(hPosys#, axis, breakpoint)

¤

Set/GetBreakpointUpdateMask

SetBreakpointUpdateMask
GetBreakpointUpdateMask

nur gültig mit POSYS® 1800/1900 falls mit Motion Control Chipsatz bestückt > V2.x

Syntax	SetBreakpointUpdateMask(hPosys, axis, breakpointID, mask) stdcall GetBreakpointUpdateMask(hPosys, axis, breakpointID) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	breakpointID	Breakpoint 1 Breakpoint 2	0 1	
	mask	unsigned 16 bits	bitmask	
Definition	SetBreakpointUpdateMask bestimmt welche Loops upgedated werden aufgrund einer Updateaktion ausgelöst durch ein Breakpoint. Falls ein bitmask für ein bestimmter Loop in *mask* gesetzt wurde, werden die Parameter für diesen Loop mit den gebufferten Werten im Register ersetzt, falls der Breakpoint aktiviert wurde und Update die definierte Breakpointaktion ist. Jed Breakpoint hat seine eigene Update Mask. Die Beschreibung ist in der folgenden Tabelle geliste			
	Name	Bit(s)	Beschreibung	
	Trajectory	0	auf 1 setzen um die Trajectory mit gebufferten Parametern upzudaten	
	Position Loop	1	auf 1 setzen um Position Loop mit gebufferten Parametern upzudaten	
	Reserved	2-15		
	z.B., falls die Update Mask für Breakpoint 1 mit 0001h konfiguriert und als Aktion Update konfiguriert wurde, werden alle Parameter für Trajectory für die spezifizierte Achse upgedated sobald Breakpoint 1 wahr wird.			
	GetBreakpointUpdateMask liefert den programmierten Zusatnd zurück			
Einschränkung				
siehe	Set/GetBreakpoint, Set/GetUpdateMask			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetBreakpointUpdateMask(hPosys#, axis, breakpointID, mask) GetBreakpointUpdateMask(hPosys#, axis, breakpointID)			

¤

Set/GetBreakpointValue

SetBreakpointValue
GetBreakpointValue

Syntax	SetBreakpointValue(hPosys, axis, breakpoint, value) stdcall GetBreakpointValue(hPosys, axis, breakpoint) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	breakpoint	Breakpoint 1 Breakpoint 2	0 1	
	value		siehe unten	

Definition

SetBreakPointValue setzt den Breakpoint Vergleichswert für die spezifizierte Achse.
Für Positions und Time Breakpoints sind dies Schwellvergleichswerte.

Der Parameterwert wird entsprechend der Triggerkondition für den ausgewählten Breakpoint interpretiert. Das Datenformat für jede Triggerbedingung ist wie folgt:

Breakpoint Trigger	Value Type	Range	Units
GreaterOrEqualCommandedPosition	signed 32 bit	-2^{31} to $2^{31}-1$	counts
LesserOrEqualCommandedPosition	signed 32 bit	-2^{31} to $2^{31}-1$	counts
GreaterOrEqualActualPosition	signed 32 bit	-2^{31} to $2^{31}-1$	counts
LesserOrEqualActualPosition	signed 32 bit	-2^{31} to $2^{31}-1$	counts
CommandedPositionCrossed	signed 32 bit	-2^{31} to $2^{31}-1$	counts
ActualPositionCrossed	signed 32 bit	-2^{31} to $2^{31}-1$	counts
Time	unsigned 32 bit	0 to $2^{32}-1$	cycles
EventStatus	2 word mask	-	Boolean status values
ActivityStatus	2 word mask	-	Boolean status values
SignalStatus	2 word mask	-	Boolean status values

Definition (Fortsetzung)

Für Level getriggerte Breakpoints ist der höherwertige Teil des Wertes die Auswahlmaske und de niederwertige Teil die Zustandsmaske. Für jedes Auswahlbit in der Auswahlmaske, welches zu 1 gesetzt ist, wird das entsprechende Bit des angegebenen Status Registers konditioniert einen Breakpoint auszulösen wenn es seinen Zustand ändert. Der Zustandsmaskenbit bestimmt, welch Status die Unterbrechung auslöst. Falls 1, dann wird das entsprechende Status Register Bit einer Break auslösen wenn auf 1 gesetzt. Falls 0, dann wird das entsprechende Status Register Bit ein Break auslösen wenn auf 0 gesetzt.

Z.B. angenommen es ist erwünscht, dass der Breakpointtyp auf "EventStatus" gesetzt ist und da ein Breakpoint erkannt wird, wann immer der Motion C omplete Bit (Bit 0 des Event Status Registers) auf 1 gesetzt wird und der Kommutierungsfehlerbit (Bit 11 des Event Status Registers auf 0 gesetzt wird. In diesem Fall würden die höher- und niederwertigen Worte des Wertes so aussehen: high = 0x801h und low = 1.

GetBreakPointValueliefert den aktuellen Breakpointwert für den spezifizierten Breakpoint.

Zwei völlig unterschiedliche Breakpoints werden unterstützt, jedes mit seinem eigenen Typ und Vergleichswert. Das Breakpointfeld spezifiziert welcher Breakpoint die Kommandos SetBreakPointValue und GetBreakPointValue adressieren.

Einschränkung

Ehe eine neue Breakpointbedingung gesetzt werden kann (Kommando SetBreakPoint), muss grundsätzlich vorher der Vergleichswert geladen werden (Kommando SetBreakPointValue). Dies ist deswegen notwendig, weil sobald die Bedingung für den Breakpoint gesetzt wurde, die POSYS® den BreakPoint Value Register verwendet und wenn bis dahin der Wert nicht entsprechend definiert wurde, wird sich der Breakpoint auch nicht wie erwartet verhalten.

siehe	Set/GetBreakpoint
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)
MotionScript® Aufruf-konvention	SetBreakpointValue(hPosys#, axis, breakpoint, value) GetBreakpointValue(hPosys#, axis, breakpoint)

¤

Set/GetInterruptMask

SetInterruptMask
GetInterruptMask

Syntax	SetInterruptMask(hPosys, axis, interruptMask) stdcall GetInterruptMask(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	InterruptMask	Motion complete Wrap-around Breakpoint 1 Capture received Motion error In positive limit In negative limit Instruction error Commutation error Breakpoint 2	0001h 0002h 0004h 0008h 0010h 0020h 0040h 0080h 0800h 4000h	
Definition	SetInterruptMaskbestimmt welche Bits im Event Status Register der spezifizierten Achse eine Interrupt auslösen. Für jedes InterruptMask Bit dass auf 1 gesetzt ist, wird das entsprechende Event Status Register Bit einen Interrupt auslösen, wenn der Status Register Bit auf "high" ges wird (gesetzt auf 1). Auf 0 gesetzte InterruptMask Bits lösen kein Interrupt aus. GetInterruptMaskliest die aktuelle Einstellung der Mask für die spezifizierte Achse. POSYS® 18xx/19xx: SetInterruptMaskkontrolliert auch CAN Event Notification wenn die POSYS® über die C AN 2 Schnittstelle betrieben wird. Immer sobald ein Hostinterrupt aktiviert wurde, wird eine C AN Message generiert mit ID 0x180 + nodeID. Betroffene C AN Nodes werden von diesem Wechs Event Status Register informiert. Beispiel: Der InterruptMask Wert 28h löst einen Interrupt aus wenn entweder der "in positive limit" Bit oder "capture received" Bit des Event Status Registers aktiv wird (auf 1 gesetzt).			
Einschränkung				
siehe	ClearInterrupt, GetInterruptAxis			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetInterruptMask(hPosys#, axis, interruptMask) GetInterruptMask(hPosys#, axis)			

¤

Kommutierung

GetPhaseCommand

GetPhaseCommand

Syntax	GetPhaseCommand(hPosys, axis, phase) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	phase	PhaseA PhaseB PhaseC	0 1 2	
Ergebnis	Name	Type		
	motor command	signed 16 bits	Range	-2^{15} to $2^{15}-1$
			Scaling	$100/2^{15}$
			Units	% output
Definition	GetPhaseCommand liefert die aktuelle Drehmomentsollwertvorgabe für die Phasen A, B und C die spezifizierte Achse. Dies sind Werte die direkt nach der Kommutierung an den Motor ausgegeben werden. Skalierungsbeispiel: Wenn der Wert -4489 zurückgeliefert wird (EE77h) für eine gegebene Achs und Phase, dann entspricht dieser Wert -4489 * 100 / 32768 = -13,7% des möglichen Ausgabewertes.			
Einschränkung	PhaseC ist nur gültig wenn der entsprechende Motortyp für 3-phasige Kommutierung ausgewählt wurde.			
siehe	InitializePhase, Set/GetNumberPhases, Set/GetMotorType			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	GetPhaseCommand(hPosys#, axis, phase)			

¤

InitializePhase

Syntax	InitializePhase(hPosys, axis) stdcall		
Motortyp		DC Brushless	
Argumente	Name	Instance	Encoding
	hPosys	cardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3
Definition	InitializePhase initialisiert die Phasenwinkel für die spezifizierte Achse und verwendet dabei entweder die algorythmische oder Hallsensor-basierte Methode, ausgewählt durch das Kommando Set/GetPhaseInitializationMode.		
Einschränkung	Warnung: Wenn die Phaseninitialisierung algorythmisch ist, dann kann es passieren, dass der Motor plötzlich in unkontrollierter Weise losfährt.		
siehe	GetPhaseCommand, Set/GetCommutationMode		
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)		
MotionScript® Aufruf-konvention	InitializePhase(hPosys#, axis)		

¤

Set/GetCommutationMode

SetCommutationMode
GetCommutationMode

Syntax	SetCommutationMode(hPosys, axis, mode) stdcall GetCommutationMode(hPosys, axis) stdcall		
Motortyp		DC Brushless	
Argumente	Name	Instance	Encoding
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3
	mode	sinusoidal hall-based microstepping	0 1 2
Definition	SetCommutationMode setzt die Art der Kommutierungsmethode fest für die spezifizierte Achse Falls "sinusoidal" und der Motor dreht sich, wird das eingehende Enkodersignal verwendet um d Phasenwinkel zu kalkulieren. Dieser Winkel wird wiederum verwendet, sinusförmig variierende Ausgaben für jede Motorwicklung zu generieren. Falls "Hall-based" werden die Hallsensoreneingänge verwendet um die Motorwicklungen zu kommutieren, dabei wird entweder die "six-step" oder "trapezoidal" Form verwendet. Falls "microstepping" wird der Ausgang des Trajectory Generators verwendet um die Phasenwin zu kalkulieren. Der Winkel wird der Reihe nach verwendet um sinusförmig variierende Ausgabe für jede Motorphase zu kalkulieren. "Microstepping" wird manchmal benutzt mit bürstenlosen Servommotoren um sie "manuell" zu bewegen ehe die Phaseninitialisierung stattgefunden hat. Alternativ kann "microstepping" mit Schrittmotoren oder AC Induktionsmotoren verwendet werden, wenn Frequenzsynthese alles is was benötigt wird um den Motor zu bewegen. GetCommutationMode liefert die eingestellte Methode zurück.		
Einschränkung			
siehe	Set/GetPhasePrescale, Set/GetPhaseCounts		
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)		
MotionScript® Aufruf-konvention	SetCommutationMode(hPosys#, axis, mode) GetCommutationMode(hPosys#, axis)		

¤

Set/GetNumberPhases

SetNumberPhases
GetNumberPhases

nur benötigt für POSYS® 8xx-B und 9xx

Syntax	SetNumberPhases(hPosys, axis, phase) stdcall GetNumberPhases(hPosys, axis) stdcall		
Motortyp		DC Brushless	Microstepping
Argumente	Name	Instance	Encoding
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3
	mode	1Phase 2Phases 3Phases	1 2 3
Definition	SetNumberPhasessetzt die Anzahl Phasen, 1, 2 oder 3, für die Kommutierung für die spezifizierte Achse. GetNumberPhasesliest die Anzahl Phasen die für die Achse gesetzt wurde.		
Einschränkung	In PWM Sign/Magnitude Ausgangssignalmodus kann die Anzahl Phasen auf entweder 1 oder 2 gesetzt werden. In PWM 50/50 Ausgangssignalmodus kann die Anzahl Phasen 1, 2 oder 3 sein.		
siehe	GetPhaseCommand, InitializePhase, Set/GetPhaseOffset, Set/GetOutputMode		
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)		
MotionScript® Aufruf-konvention	SetNumberPhases(hPosys#, axis, phase) GetNumberPhases(hPosys#, axis)		

¤

Set/GetPhaseAngle

SetPhaseAngle
GetPhaseAngle

Syntax	SetPhaseAngle(hPosys, axis, angle) stdcall GetPhaseAngle(hPosys, axis) stdcall			
Motortyp		DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	angle	unsigned 16 bits	Range	0 to 2^{15}-1
			Scaling	unity
			Units	counts microsteps
Definition	SetPhaseAngle setzt den Kommutierungswinkel für die spezifizierte Achse sofort. GetPhaseAngle liest den programmierten Wert für die aktuelle Einstellung. Um Enkodereinheite in den aktuellen Phasenwinkel zu konvertieren, teilen Sie die Anzahl Enkodersignale pro elektrischem Zyklus und multiplizieren Sie mit 360. z.B. falls ein Wert von 500 mit dem Kommando GetPhaseAngle empfangen wurde, und die Anza Enkodersignale pro elektrischem Zyklus wurde auf 2000 gesetzt, (SetPhaseC ounts Kommando entspricht dies dann einem Winkel von: (500/2000)*360 = 90 Grad für die aktuelle Phasenwinkelposition.			
Einschränkung	Der spezifizierte Winkel darf nicht die maximale Anzahl Enkodersignale pro elektrischem Zyklus überschreiten, die mit dem Kommando SetPhaseCounts gesetzt wurde.			
siehe	Set/GetPhaseCounts, InitializePhase, Set/GetMotorType, Set/GetNumberPhases			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetPhaseAngle(hPosys#, axis, angle) GetPhaseAngle(hPosys#, axis)			

¤

Set/GetPhaseCorrectionMode

SetPhaseCorrectionMode GetPhaseCorrectionMode

Syntax	SetPhaseCorrectionMode(hPosys, axis, mode) stdcall GetPhaseCorrectionMode(hPosys, axis) stdcall		
Motortyp		DC Brushless	
Argumente	Name	Instance	Encoding
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3
	mode	disable enable	0 1
Definition	SetPhaseCorrectionMode setzt den Modus für die Phasenkorrektur für die spezifizierte Achse a entweder aktiviert (0) oder deaktiviert (1). Wenn die Phasenkorrektur aktiviert ist, wird das Enkodersignal verwendet um den Kommutierungsphasenwinkel mit jeder Umdrehung des Motor upzudaten. Dies stellt sicher, dass der Kommutierungswinkel korrekt bleibt, obwohl einige Enkodersignale z.B. wegen Leitungsstörungen verloren gegangen sein könnten, oder weil die Anzahl Enkodersignale pro elektrischer Phase kein Integerwert darstellt. GetPhaseCorrectionMode zeigt den aktuellen Modus an.		
Einschränkung			
siehe	GetPhaseCommand, InitializePhase, Set/GetMotorType, Set/GetNumberPhases, Set/GetPhaseCounts		
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)		
MotionScript® Aufruf-konvention	SetPhaseCorrectionMode(hPosys#, axis, mode) GetPhaseCorrectionMode(hPosys#, axis)		

¤

Set/GetPhaseCounts

SetPhaseCounts
GetPhaseCounts

Syntax	SetPhaseCounts(hPosys, axis, counts) stdcall GetPhaseCounts(hPosys, axis) stdcall		
Motortyp		DC Brushless	Microstepping
Argumente	Name	Instance	Encoding
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3
		Type	
	counts	unsigned 32 bits	POSYS® 18xx/19xx:
			Range: 1 to 2^{15}-1
			Scaling: unity
			Units: counts steps
			POSYS® 8xx-B/9xx:
			Range: 0 to 2^{15}-1
			Scaling: unity
			Units: counts steps
Definition	Für Achsen die als bürstenlose Servomotorachsen konfiguriert wurden, bestimmt SetPhaseCounts die Anzahl Inkremente pro elektrischem Zyklus des Motors. Die Anzahl elektrischer Zyklen ist gleich ½ der Anzahl Pole (nicht Polpaare). Falls dieser Wert kein Integer ist, sollte der nächste Integerwert und der Phasenkorrekturmodus (SetPhaseC orrectionMode) verwendet werden. Für Achsen die für den Mikroschrittmodus konfiguriert wurden, wird mit dem Kommando SetPhaseCountsdie Anzahl Mikroschritte pro Vollschritt definiert. Der Parameter der für dieses Kommando Anwendung findet, definiert die Anzahl Mikroschritte pro elektrischen Zyklus (4x die gewünschte Anzahl Mikroschritte). z.B. um 64 Mikroschritte pro Vollschritt zu definieren wird der Parameterwert 256 übergeben. Die maximale Anzahl Mikroschritte pro Vollschritt die angegeben werden kann ist 256 und der dazugehörige Parameter ist dann 1024. GetPhaseCountsliefert den programmierten Wert zurück.		
Einschränkung			
siehe	Set/GetPhaseAngle		
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)		
MotionScript® Aufruf-konvention	SetPhaseCounts(hPosys#, axis, counts) GetPhaseCounts(hPosys#, axis)		

¤

Set/GetPhaseInitializeMode

SetPhaseInitializeMode
GetPhaseInitializeMode

Syntax	SetPhaseInitializeMode(hPosys, axis, mode) stdcall GetPhaseInitializeMode(hPosys, axis) stdcall		
Motortyp		DC Brushless	
Argumente	Name	Instance	Encoding
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3
	mode	Algorithmic Hall-based	0 1
Definition	SetPhaseInitializeMode definiert die Methode auf welche Art die Kommutierung für die spezifizierte Achse stattfindet. Folgende Optionen sind möglich: algorythmic und Hall-based. Die algorythmische Methode stimuliert nur kurz die Motorwicklungen und aufgrund der Rückmeldungen, stellt die POSYS® die Position der Phasen fest. In der Hall-basierten Methode werden die 3 Hallsensoreingänge verwendet um die Phasen zu bestimmen. GetPhaseInitializeMode liest welche Methode eingestellt ist.		
Einschränkung	Die algorythmische Methode sollte nur verwendet werden, wenn die Achse in beide Richtungen freibeweglich und eine kurze, unkontrollierte Bewegung auf der Achse möglich ist, ohne evtl. Schäden an der Achse, Mechanik etc. zu riskieren.		
siehe	GetPhaseCommand, InitializePhase, Set/GetMotorType, Set/GetNumberPhases		
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)		
MotionScript® Aufruf-konvention	SetPhaseInitializeMode(hPosys#, axis, mode) GetPhaseInitializeMode(hPosys#, axis)		

¤

© POSYS Motion Control GmbH & Co.KG, 2013

SetPhaseInitializeTime
GetPhaseInitializeTime

Syntax	SetPhaseInitializeTime(hPosys, axis, time) stdcall GetPhaseInitializeTime(hPosys, axis) stdcall			
Motortyp		DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	time	unsigned 16 bits	Range	0 to 2^{15}-1
			Scaling	unity
			Units	cycles
Definition	SetPhaseInitializeTime definiert die Zeitkonstante (nach Zyklen) für die Algorythmic Phase Initialization. Dieser Wert bestimmt die Dauer für jedes der 4 einzelnen Segmente für die alogrythmische Initialisierung der Phasen. Für Details über die algorytmische Initialisierung der Phasen, lesen Sie bitte auch im Benutzer-Handbuch nach. GetPhaseInitializeTime liest den programmierten Wert zurück.			
Einschränkung				
siehe	GetPhaseCommand, InitializePhase, Set/GetNumberPhases, Set/GetPhaseInitializeMode, Set/GetMotorType			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetPhaseInitializeTime(hPosys#, axis, time) GetPhaseInitializeTime(hPosys#, axis)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

SetPhaseOffset
GetPhaseOffset

Syntax	SetPhaseOffset(hPosys, axis, offset) stdcall GetPhaseOffset(hPosys, axis) stdcall			
Motortyp		DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	time	unsigned 16 bits	POSYS® 18xx/19xx:	
			Range	-1 to 2^{15}-1
			Scaling	unity
			Units	counts
			POSYS® 8xx-B/9xx:	
			Range	0 to 2^{15}-1
			Scaling	unity
			Units	counts
Ergebnis				
Definition	SetPhaseOffsetsetzt den Offset der Indexmarkierung der spezifizierten Achse bis zum maximalen Ausgabewert der Phase A. Dieses Kommando hat keinen sofortigen Effekt auf den Kommutierungswinkel, sondern erst wenn der Indextakt erfasst wurde. GetPhaseOffsetliest den aktuellen Wert für den Offset der Phase. Um Enkodersignale eines Phasenwinkels in Grad zu konvertieren, teilen Sie die Anzahl Enkodersignale pro elektrischem Zyklus und multiplizieren Sie den Wert mit 360. z.B. falls ein W von 500 mit dem Kommando SetPhaseOffset spezifiziert wurde und die Anzahl Signale pro elektrischem Zyklus wurde auf 2000 (Kommando SetPhaseC ounts) gesetzt, entspricht dies ein Winkel von: (500/2000)*360 = 90 Grad Phasenwinkel an der Indexmarkierung.			
Einschränkung	Ehe der erste Indextakt erfasst wurde, liefert das Kommando GetPhaseOffset den Wert 65535 zurück.			
siehe	GetPhaseCommand, InitializePhase, Set/GetNumberPhases, Set/GetMotorType			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetPhaseOffset(hPosys#, axis, offset) GetPhaseOffset(hPosys#, axis)			

¤

Set/GetPhasePrescale

SetPhasePrescale
GetPhasePrescale

Syntax	SetPhasePrescale(hPosys, axis, scale) stdcall GetPhasePrescale(hPosys, axis) stdcall		
Motortyp		DC Brushless	
Argumente	Name	Instance	Encoding
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3
	scale	Off 1/64 1/128 1/256	0 1 2 (nur POSYS® 18xx/19xx) 3 (nur POSYS® 18xx/19xx)
Ergebnis			
Definition	SetPhasePrescale kontrolliert die Skalierung von Inkrementen ehe sie für die Kalkulation des Kommutierungswinkels für die spezifizierte Achse verwendet werden. Wenn der Prescalemodus angewandt wird, kann die POSYS® Motoren mit einer hohen Anzahl Enkodersignale pro elektrischem Zyklus, kommutieren, z.B. Motoren mit sehr hochauflösenden Enkodern. SetPhasePrescale Off entfernt den Skalierungsfaktor. GetPhasePrescale liest die aktuell eingestellte Skalierungsmethode.		
Einschränkung			
siehe	GetPhaseCommand, InitializePhase, Set/GetNumberPhases, Set/GetMotorType		
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)		
MotionScript® Aufruf-konvention	SetPhasePrescale(hPosys#, axis, scale) GetPhasePrescale(hPosys#, axis)		

¤

© POSYS Motion Control GmbH & Co.KG, 2013

ClearPositionError

ClearPositionError

Syntax	ClearPositionError(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
Definition	ClearPositionError setzt die gegenwärtige Sollposition gleich der aktuellen Position (Enkoderposition), dabei wird der aktuelle Schleppabstandsfehler für die spezifizierte Achse auf Null gesetzt. Dieses Kommando kann verwendet werden, wenn die Achse stillsteht oder während der Bewegung. Wenn es während der Bewegung verwendet wird, sollte sich der Host im Klaren darüber sein, dass die Zielposition (im trapezförmigen und S-Kurven Modus) durch dieses Kommando nicht verändert wird.			
Einschränkung	ClearPositionErrorist ein gebuffertes Kommando. Der neue Wert wird erst durch ein Update o MultiUpdate Kommando effektiv. Das Kommando sollte nicht ausgeführt werden, während der C ontroller im S-Kurven Modus ein Bewegung ausführt.			
siehe	GetPositionError, Set/GetPositionErrorLimit, Update, MultiUpdate			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	ClearPositionError(hPosys#,axis)			

¤

DACOutputs

DACOutputsOn
DACOutputsOff

Syntax	DACOutputsOn(hPosys) stdcall DACOutputsOff(hPosys) stdcall		
Motortyp	DC Brush	DC Brushless	Microstepping
Argumente	Name	Instance	Encoding
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice
Definition	DACOutputsOnschaltet alle ±10 V optischen Schalter für die analoge Motorsignalausgabe ON. DACOutputsOffschaltet alle ±10 V optischen Schalter für die analoge Motorsignalausgabe OFF. Die Ausgänge sind mit GND verbunden. Dies entspricht auch dem Status nach dem Einschalten, einem HardReset und einem externen Reset. Das heisst sie sind OFF.		
Einschränkung	Dieses Kommando bezieht sich nur auf die analoge Motorbefehlsausgabe. Es hat keinen Effekt auf die PWM-Ausgänge. Die PWM-Ausgänge sind mit anderen Anschlüssen verbunden.		
siehe	HardReset/MCReset		
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)		
MotionScript® Aufruf-konvention	DACOutputsOn(hPosys#) DACOutputsOff(hPosys#)		

¤

GetAmpEnable

Syntax	GetAmpEnable(hPosys) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
Ergebnis	Name	Type	Range	0 - 15
	enabled1 enabled2 enabled3 enabled4 none	signed 16 bits	Returned Value	1 2 4 8 0
Definition	GetAmpEnable liefert den Status der vier Ausgänge für die Verstärkerfreigabe zurück.			
Einschränkung				
siehe	SetAmpDisable, SetAmpEnable			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	GetAmpEnable(hPosys#)			

¤

GetDerivative

Syntax	GetDerivative(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
Ergebnis	Name	Type		
	derivative	signed 16 bits	Range	-2^{15} to $2^{15}-1$
			Scaling	unity
			Units	counts/cycle
Definition	GetDerivative liefert den Derivativeanteil des aktuellen Schleppabstandsfehlers, der vom Servofilter kalkuliert wurde. Der Derivativewert wird definiert aus dem vorhergehenden Schleppabstandsfehler abzüglich den gegenwärtigen Schleppabstandsfehler. Siehe SetDerivativeTime für Details um die Derivative Sampling Time zu setzen.			
Einschränkung	Dieser Wert ist nur verfügbar wenn die POSYS® in Closed-Loop-Mode betrieben wird.			
siehe	GetIntegral, Set/GetDerivativeTime			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	GetDerivative(hPosys#, axis)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

GetIntegral

Syntax	GetIntegral(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
Ergebnis	Name	Type		
	integral	signed 32 bits	Range	-2^{31} to $2^{31}-1$
			Scaling	$1/2^{8}$
			Units	counts*cycle
Definition	GetIntegralliefert den aktuellen integralen Positionsfehler des Servofilters für die spezifizierte Achse. GetIntegral kann für die Überwachung der Masse einer Achse verwendet werden, weil Änderungen in der Masse reflektieren den Wert vom Integration Limit. Skalierungsbeispiel: Falls ein konstanter Positionsfehler von 100 Inkrementen für die Dauer von 256 Zyklen auftritt, dann ist der totale kummulierte Integralwert 100 (100*(256/256). Alternativ zeigt ein Wert von 1000 einen gespeicherten Wert von 256000 counts*cycle (1000*256).			
Einschränkung	Der integrale Positionsfehler ist nur verfügbar wenn die POSYS® im Closed-Loop-Mode betrieben wird (SetMotorMode-Kommando).			
siehe	GetDerivative, Set/GetIntegrationLimit, GetPositionError			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	GetIntegral(hPosys#, axis)			

¤

GetPositionError

Syntax	GetPositionError(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
Ergebnis	Name	Type		
	position error	signed 32 bits	Range	-2^{31} to $2^{31}-1$
			Scaling	unity
			Units	counts steps
Definition	GetPositionErrorliefert den aktuellen Schleppabstandsfehler für die spezifizierte Achse. Der Fehler ist die Differenz zwischen der aktuellen Enkoderposition und der Sollposition der vom Re vorgegeben wird. Wenn der ausgewählte Motortyp Mikroschritt oder Takt & Richtung ist, wird die Differenz bestim als die Differenz zwischen Enkoderposition (dargestellt in Schritten) und der Sollposition.			
Einschränkung				
siehe	Set/GetPosition, Set/GetPositionErrorLimit			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	GetPositionError(hPosys#, axis)			

¤

GetPositionLoopValue

nur gültig mit POSYS® 1800/1900 falls mit Motion Control Chipsatz bestückt > V2.x

Syntax	GetPositionLLoop(hPosys, axis, node) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	node	Integrator Sum Integral Contribution Deivative Biquad1 Input Biqaud2 Input	0 1 2 3 4	
Ergebnis		Type	Range/Scaling	
	value	signed 32 bits	see below	
Definition	GetPositionLoopValue wird verwendet um den Wert eines Nodes in der Regelschleife zu finden Das Benutzerhandbuch liefert hierzu weiter Informationen. Dieses Kommando verwendet imme 32-Bit Daten, aber der Bereich und das Format variieren abhängig vom node wie folgt.			
	Parameter	Range	Scaling	Units
	Integrator Sum	-2^{31} to $2^{31}-1$	unity	(counts or microsteps)*cycles
	Integral Contribution	-2^{31} to $2^{31}-1$	$100*Kout/(2^{16})$	
	Derivative	-2^{15} to $2^{15}-1$	unity	(counts oder microsteps)/cycles
	Biquad1 Input	-2^{15} to $2^{15}-1$	unity	counts or microsteps
	Biquad2 Input	-2^{15} to $2^{15}-1$	unity	counts or microsteps
Einschränkung				
siehe	Set/GetPositionLoop			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	GetPositionLoopValue(hPosys#, axis, node)			

¤

SetAmpDisable

Syntax	SetAmpDisable(hPosys, disable) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
		disable1	1	
		disable2	2	
		disable3	4	
		disable4	8	
Definition	SetAmpDisable kann verwendet werden einen, oder jede Kombination oder alle Verstärkerfreigabeausgänge zu deaktivieren. Nach einem Power-On oder einem externen Reset oder HardReset Kommando befinden sich alle Verstärkerfreigabeausgänge im AUS-Zustand (low). Das Setzen eines Verstärkerfreigabeausganges versetzt ihn in den EIN-Zustand (high). Das SoftReset Kommando hat keinen Effekt auf die Verstärkerfreigaben. Das Kommando SetAmpDisable schaltet die Verstärkerfreigaben AUS.			
Einschränkung				
siehe	SetAmpEnable, GetAmpEnable			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetAmpDisable(hPosys#, disable)			

¤

SetAmpEnable

Syntax	SetAmpEnable(hPosys, enable) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
		enable1 enable2 enable3 enable4	1 2 4 8	
Definition	SetAmpEnable kann verwendet werden einen, oder jede Kombination oder alle Verstärkerfreigabeausgänge zu aktivieren. Nach einem Power-On oder einem externen Reset oder HardReset Kommando befinden sich alle Verstärkerfreigabeausgänge im AUS-Zustand (low). Das Setzen eines Verstärkerfreigabeausganges versetzt ihn in den EIN-Zustand (high). Das SoftReset Kommando hat keinen Effekt auf die Verstärkerfreigaben. Das Kommando SetAmpEnable schaltet die Verstärkerfreigaben EIN.			
Einschränkung	Das SoftReset Kommando hat keinen Effekt auf die Verstärkerfreigaben.			
siehe	SetAmpDisable, GetAmpEnable			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetAmpEnable(hPosys#, enable)			

¤

Set/GetAutoStopMode

SetAutoStopMode
GetAutoStopMode

Syntax	SetAutoStopMode(hPosys, axis, mode) stdcall GetAutoStopMode(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	mode	Disable Enable	0 1	
Definition	SetAutoStopModebestimmt das Verhalten der spezifizierten Achse wenn ein Fehler auftritt. Fal automatischer Stopp aktiviert ist (SetAutoStopMode(enable)), schaltet die Achse bei einem Fehl in den Open-Loop-Modus. Falls der automatische Stopp deaktiviert ist (SetAutoStopMode(disabl reagiert die Achse nicht auf einen Fehler. GetAutoStopModeliefert den aktuell programmierten Zustand zurück.			
Einschränkung	Wenn die Einstellung „SetEncoderSource none“ verwendet wird, wird das Setzen des „AutoStopMode“ auf „Enable“ im Falle eines Schleppabstandsfehlers den Motor nicht stoppen.			
siehe	GetEventStatus, Set/GetPositionErrorLimit			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetAutoStopMode(hPosys#, axis, mode) GetAutoStopMode(hPosys#, axis)			

¤

Set/GetAuxiliaryEncoderSource

SetAuxiliaryEncoderSource
GetAuxiliaryEncoderSource

nur gültig für POSYS® 18xx/19xx Serien

Syntax	SetAuxiliaryEncoderSource(hPosys, axis, mode, auxiliaryAxis) stdcall GetAuxiliaryEncoderSource(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	mode	disable enable	0 1	
	auxiliaryAxis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
Definition	SetAuxiliaryEncoderSource kontrolliert die Dual-Loop-Funktionalität. Der „mode“-Parameter aktiviert oder deaktiviert den zweiten Enkoder für „axis“. Der „auxiliaryaxis“ -Parameter definie welcher Enkodereingang als Zusatzterm für „axis“ verwendet werden soll. Um die aktuelle Position des AuxiliaryEncoders zu bestimmen verwenden Sie GetActualPosition(auxiliaryAxis). GetAuxiliaryEncoderSource liefert die augenblickliche Einstellung zurück.			
Einschränkung	Diese Kommando ist nur für die POSYS® 18xx/19xx verfügbar. Um die aktuelle Position des zweiten Enkoders zu bestimmen, verwenden Sie das Kommando GetActualPosition (auxiliaryAxis). Um instabile Systeme zu vermeiden, sollte immer der zweite Enoder (AuxiliaryEncoder) mindes die gleiche oder eine bessere Auflösung haben als der Hauptenkoder. Dieses Kommando ist nicht mehr verfügbar in POSYS® 1800/1900 C ontrollern mit Piezo-Funktionalität.			
siehe				
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetAuxiliaryEncoderSource(hPosys#, axis, mode, auxiliaryAxis) GetAuxiliaryEncoderSource(hPosys#, axis)			

¤

Set/GetBiQuadCoefficient

SetBiQuadCoefficient
GetBiQuadCoefficient

nur verfügbar für POSYS® 18xx/19xx Series

Syntax	SetBiQuadCoefficient(hPosys, axis, coefficientID, filterID, value) stdcall GetBiQuadCoefficient(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	coefficientID	FilterEnable coefficient B0 coefficient B1 coefficient B2 coefficient A1 coefficient A2 coefficient k	0 1 2 3 4 5 6	
	filterID	Filter0 Filter1	0 1	
		Type		
	value	signed 16 bit	Range	-2^{15} to $2^{15}-1$
			Scaling	unity
Definition	SetBiQuadCoefficient setzt den ausgewählten Koeffizienten für den ausgewählten Bi-Quad Output Filter der ausgewählten Achse. Der Filter einer Achse wird aktiviert durch Senden einer coefficientID 0 (FilterEnable) und einem value 1; der value 0 deaktiviert den Filter. Die Ausgabe des Filters zu einer gegebenen Zeit n (Y[n]) wird bestimmt von Eingabedaten (X[n] durch folgende Formel: Y[n] = K*(B0*X[n] + B1*X[n-1] + B2*X[n-2] + A1*Y[n-1] + A2*Y[n-2] B0, B1, B2, A1 und A2 sind Vorzeichen-abhängige Koeffizienten; K ist ein 15-Bit positiver Skalie um Rundungsfehler zu reduzieren. GetBiQuadCoefficient liefert den Koeffizienten für den ausgewählten Bi-Quad Ausgangsfilter fü die gewählte Achse.			
Einschränkung	Jede POSYS® 18xx/19xx unterstützt bis zu 2 Bi-Quad Filter pro Achse (dies ist abhängig von de Anzahl durch die Karte unterstützte Achsen). Falls beide Filter aktiviert wurden, erfolgen ihre Ausgaben nacheinander (filter0 gefolgt von filter1). Falls Filter0 für eine Achse deaktiviert ist, ist auch filter1 auf der Achse deaktiviert, egal welche Einstellungen für filter1 gelten. Während der Änderung von Filterparamtern ist es von Vorteil die Filter zu deaktivieren und erst nach dem Setzen der Parameter wieder zu aktivieren. Dies verhindert unerwartete plötzliche Änderungen im System (Motorausgabewerte). Dieses Kommando ist nicht verfügbar für die POSYS® 700/800/800-B/900 Serien. Es ist nur gül mit die POSYS® 1800/1900 Serien. Dieses Kommando ist nicht mehr verfügbar in POSYS® 1800/1900 C ontrollern mit Piezo-Funktionalität.			
siehe				
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetBiQuadCoefficient(hPosys#, axis, coefficientID, filterID, value) GetBiQuadCoefficient(hPosys#, axis)			

¤

Set/GetDerivativeTime

SetDerivativeTime
GetDerivativeTime

Syntax	SetDerivativeTime(hPosys, axis, time) stdcall GetDerivativeTime(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	time	unsigned 16 bits	Range	0 to 2^{15}-1
			Scaling	unity
			Units	cycles
Definition	SetDerivativeTime	setzt den Derivativeperiodenteil für den Servofilter nach Anzahl Zyklen um den Derivativeanteil für die spezifizierte Achse zu kalkulieren.		
	GetDerivativeTime	liest den programmierten Wert zurück.		
Einschränkung	Diese Kommando ist gebuffert. Der neue Wert wird erst nach dem Senden eines Update- oder MultiUpdate-Kommandos an die POSYS® aktuell. Dieses Kommando hat keinen Einfluss auf die eigentliche Servoloopupdaterate der POSYS®. Di Servoloopupdaterate wird mit dem Kommando SetSamplingTime gesetzt.			
siehe	GetDerivative			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetDerivativeTime(hPosys#, axis, time) GetDerivativeTime(hPosys#, axis)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

SetIntegrationLimit
GetIntegrationLimit

Syntax	SetIntegrationLimit(hPosys,axis, limit) stdcall GetIntegrationLimit(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
Ergebnis	Name	Type		
	limit	unsigned 32 bits	Range	0 to 2^{31}-1
			Scaling	$1/2^{8}$
			Units	counts*cycle
Definition	SetIntegrationLimit lädt den Integralanteil in den Servofilter für die spezifizierte Achse. GetIntegrationLimit liest den programmierten Wert zurück. Skalierungsbeispiel: Die Skalierung ist dieselbe wie für das Kommando GetIntegral. Ein konstan Positionsfehler von 100 Zählern über eine Periode von 256 Zyklen würde einen Intergralwert vo 100(100*256/256) zum Ergebnis haben, und daher würde ein IntegralLimit Wert von 100 den totalen, akkumulierten Integralfehler auf 25600 count*cycles begrenzen.			
Einschränkung	Dies ist ein gebuffertes Kommando und wird erst durch ein Update oder MultiUpdate Kommando aktiviert.			
siehe	GetIntegral, GetDerivative, Set/GetDerivativeTime, MultiUpdate, Update			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetIntegrationLimit(hPosys#,axis, limit) GetIntegrationLimit(hPosys#, axis)			

¤

Set/GetKaff

SetKaff
GetKaff

Syntax	SetKaff(hPosys, axis, Kaff) stdcall GetKaff(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	Kaff	unsigned 16 bits	Range	0 to 2^{15}-1
			Scaling	unity
Definition	SetKaff setzt die Acceleration Feedforward Verstärkung des digitalen Servofilters für die spezifizierte Achse. GetKaff liest den programmierten Wert für die Acceleration Feedforward Verstärkung.			
Einschränkung	SetKaff ist ein gebuffertes Kommando und wird erst nach einem Update oder MultiUpdate Kommando aktiviert.			
siehe	Set/GetKd, Set/GetKi, Set/GetKout, Set/GetKp, Set/GetKvff, MultiUpdate, Update			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetKaff(hPosys#, axis, Kaff) GetKaff(hPosys#, axis)			

¤

Set/GetKd

SetKd
GetKd

Syntax	SetKd(hPosys, axis, Kd) stdcall GetKd(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	Kd	unsigned 16 bits	Range	0 to 2^{15}-1
			Scaling	unity
Definition	SetKd setzt die Derivative Verstärkung für den digitalen Servofilter für die spezifizierte Achse. GetKd liest den programmierten Wert für die Derivative Verstärkung.			
Einschränkung	SetKd ist ein gebuffertes Kommando und wird erst nach einem Update oder MultiUpdate Kommando aktiviert.			
siehe	Set/GetKi, Set/GetKout, Set/GetKp, Set/GetKaff, Set/GetKvff, MultiUpdate, Update			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetKd(hPosys#, axis, Kd) GetKd(hPosys#, axis)			

¤

Set/GetKi

SetKi
GetKi

Syntax	SetKi(hPosys, axis, Ki) stdcall GetKi(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	Ki	unsigned 16 bits	Range	0 to 2^{15}-1
			Scaling	unity
Definition	SetKi setzt die Integralverstärkung für den digitalen Servofilter für die spezifizierte Achse. GetKi liest den programmierten Wert für die Integralverstärkung.			
Einschränkung	SetKi ist ein gebuffertes Kommando und wird erst nach einem Update oder MultiUpdate Kommando aktiviert.			
siehe	Set/GetKd, Set/GetKout, Set/GetKp, Set/GetKaff, Set/GetKvff, MultiUpdate, Update			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetKi(hPosys#, axis, Ki) GetKi(hPosys#, axis)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

SetKout
GetKout

Syntax	SetKout(hPosys,axis, Kout) stdcall GetKout(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	Kout	unsigned 16 bits	Range	0 to 2^{16}-1
			Scaling	$100/2^{16}$
			Units	% output
Definition	SetKout setzt den Skalierungsfaktor für das Ausgangssignal des digitalen Servofilters für die spezifizierte Achse. Der Standardwert von Kout ist 65536. GetKout liest den programmierten Wert des Skalierungsfaktors für das Ausgangssignal. Beispiel: Um die Skalierung auf die Hälfte zu setzen, muss für Kout 32767 eingegeben werden.			
Einschränkung	Dieses Kommando ist NICHT gebuffert. Es wird sofort aktiv nachdem es gesendet wurde.			
siehe	Set/GetKd, Set/GetKi, Set/GetKp, Set/GetKaff, Set/GetKvff, MultiUpdate, Update			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetKout(hPosys#, axis, Kout) GetKout(hPosys#, axis)			

¤

Set/GetKp

SetKp
GetKp

Syntax	SetKp(hPosys, axis, Kp) stdcall GetKp(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	Kp	unsigned 16 bits	Range	0 to 2^{15}-1
			Scaling	unity
Definition	SetKp setzt die proportionale Verstärkung des digitalen Servofilters für die spezifizierte Achse. GetKp liest den programmierten Wert für die proportionale Verstärkung.			
Einschränkung	SetKp ist ein gebuffertes Kommando und wird erst nach einem Update bzw. MultiUpdate Kommando aktiviert.			
siehe	Set/GetKd, Set/GetKi, Set/GetKout, Set/GetKaff, Set/GetKvff, MultiUpdate, Update			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetKp(hPosys#, axis, Kp) GetKp(hPosys#, axis)			

¤

Set/GetKvff

SetKvff
GetKvff

Syntax	SetKvff(hPosys, axis, Kvff) stdcall GetKvff(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	Kvff	unsigned 16 bits	Range	0 to 2^{15}-1
			Scaling	unity
Definition	SetKvffsetzt die Velocity Feedforward Verstärkung des digitalen Servofilters für die spezifiziert Achse. GetKvffliest den programmierten Wert für die Velocity Feedforward Verstärkung.			
Einschränkung	SetKvff ist ein gebuffertes Kommando und wird erst nach einem Update bzw. MultiUpdate Kommando aktiviert.			
siehe	Set/GetKd, Set/GetKi, Set/GetKout, Set/GetKp, Set/GetKaff, MultiUpdate, Update			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetKvff(hPosys#, axis, Kvff) GetKvff(hPosys#, axis)			

¤

Set/GetMotorLimit

SetMotorLimit
GetMotorLimit

Syntax	SetMotorLimit(hPosys, axis, limit) stdcall GetMotorLimit(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	limit	unsigned 16 bits	Range	0 to 2^{15}-1
			Scaling	$100/2^{15}$
			Units	% output
Definition	SetMotorLimitsetzt die maximale Ausgabespannung für das Ausgangssignal des digitalen Servofilters der spezifizierten Achse. Ausgangssignalwerte jenseits dieses Wertes werden abgeschnitten. z.B. falls die Begrenzung auf 1000 gesetzt wurde und der Servofilter ermittelt, d der Ausgabewert 1100 sein sollte, wird der aktuelle Wert der ausgegeben wird 1000 sein. Umgekehrt, falls der ermittelte Wert -1100 sein soll, dann wird der ausgegebene Wert -1000 sei Diese Kommando ist nützlich um Verstärker, Motoren oder andere Systemkomponenten vor Schaden zu schützen, wenn man weiss, dass höhere Werte unverträglich für das System sind. GetMotorLimitliest den aktuellen Wert für die Motorstromausgabegrenzung. Skalierungsbeispiel: Wenn die Begrenzung auf 75% der vollen Skala begrenzt werden soll, dann sollte dieses Regist mit folgendem Wert geladen werden: 75,0*32768/100 = 24,576 (dezimal). Dies entspricht eine Hexadezimalwert von 06000h.			
Einschränkung	Diese Kommando hat nur im Closed-Loop Modus Einfluss auf das Motorausgangssignal. Im Open-Loop Modus hat dieses Kommando keinen Effekt.			
siehe	Set/GetMotorBias, Set/GetMotorCommand, Set/GetMotorMode			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetMotorLimit(hPosys#, axis, limit) GetMotorLimit(hPosys#, axis)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

SetPositionErrorLimit
GetPositionErrorLimit

Syntax	SetPositionErrorLimit(hPosys, axis, limit) stdcall GetPositionErrorLimit(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	limit	unsigned 32 bits	Range	0 to 2^{31}-1
			Scaling	unity
			Units	counts
Definition	SetPositionErrorLimitsetzt den absoluten Wert für den maximal erlaubbaren Schleppabstandsfehler für die spezifizierte Achse. Bei Überschreitung des Schleppabstandsfehle wird ein Bewegungsfehler (motion error) ausgelöst. Ein solcher Bewegungsfehler kann, je nach mit dem Kommando SetAutoStopMode definierten Einstellung, einen Stopp veranlassen oder nic GetErrorPositionLimitliest den programmierten Wert zurück.			
Einschränkung				
siehe	GetPositionError, Set/GetActualPosition, Set/GetPosition			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetPositionErrorLimit(hPosys#, axis, limit) GetPositionErrorLimit(hPosys#, axis)			

¤

Set/GetPositionLoop

SetPositionLoop
GetPositionLoop

nur gültig mit POSYS® 1800/1900 falls mit Motion Control Chipsatz bestückt > V2.x

Syntax	SetPositionLoop(hPosys, axis, parameter, value) stdcall GetPositionLoop(hPosys, axis, parameter) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
				Encoding
	axis	Axis1		0
		Axis2		1
		Axis3		2
		Axis4		3
	parameter	PID Proportional Gain (Kp)		0
		PID Integrator Gain (Ki)		1
		PID Integrator Limit (Ilimit)		2
		PID Derivative Gain (Kd)		3
		PID Derivate Time		4
		PID Output Gain (Kout)		5
		Velocity Feedforward Gain (Kvff)		6
		Acceleration Feedforward Gain (Kaff)		7
		Biquad1, enable filter		8
		Biquad1, coefficientB0		9
		Biquad1, coefficientB1		10
		Biquad1, coefficientB2		11
		Biquad1, coefficientA1		12
		Biquad1, coefficientA2		13
		Biquad1, coefficientK		14
		Biquad2, enable filter		15
		Biquad2, coefficientB0		16
		Biquad2, coefficientB1		17
		Biquad2, coefficientB2		18
		Biquad2, coefficientA1		19
		Biquad2, coefficientA2		20
		Biquad2, coefficientK		21
		Type		Range/Scaling
	value	signed 32 bits		see below
Definition	SetPositionLoopsetzt die Parameter für die Regelung (PID). Im Benutzerhandbuch ist detaillierter Beschrieben wie jeder Parameter wirkt und wie es benutzt wird. Dieses Kommando verwendet immer 32-Bit Daten, aber der Bereich und das Format variieren abhängig vom *parameter*wie in der Tabelle beschrieben. GetPositionLoopliest den programmierten Wert zurück.			

Parameter	Range	Scaling	Units
Velocity Feedforward Gain (Kvff)	0 to 2^{15-1}	unity	
Acceleration Feedforward Gain (Kaff)	0 to 2^{15-1}	unity	
PID Proportional Gain (Kp)	0 to 2^{15-1}	unity	
PID Integrator Gain (Ki)	0 to 2^{15-1}	1/256	
PID Derivative Gain (Kd)	0 to 2^{15-1}	unity	

PID Integrator Limit (Limit)	0 to 2^{31-1}	unity	count*cycles
PID Derivative Time	0 to 2^{15-1}	unity	cycles
PID Output Gain (Kout)	0 to 2^{16-1}	$100/2^{16}$	%output
Biquad1, Enable Filter	0 to 1	0=disable 1=enable	
Biquad1, coefficientB0	-2^{15} to 2^{15-1}	unity	
Biquad1, coefficientB1	-2^{15} to 2^{15-1}	unity	
Biquad2, coefficientB2	-2^{15} to 2^{15-1}	unity	
Biquad1, coefficientA1	-2^{15} to 2^{15-1}	unity	
Biquad1, coefficientA2	-2^{15} to 2^{15-1}	unity	
Biquad1, coefficientK	0 to 2^{15-1}	unity	
Biquad2, Enable Filter	0 to 1	0=disable 1=enable	
Biquad2, coefficientB0	-2^{15} to 2^{15-1}	unity	
Biquad2, coefficientB1	-2^{15} to 2^{15-1}	unity	
Biquad2, coefficientB2	-2^{15} to 2^{15-1}	unity	
Biquad2, coefficientA1	-2^{15} to 2^{15-1}	unity	
Biquad2, coefficientA2	-2^{15} to 2^{15-1}	unity	
Biquad2, coefficientK	0 to 2^{15-1}	unity	

Viele dieser Parameter sind selbsterklärend. Jedoch werden im Folgenden einige Zusatzinformationen bereitgestellt zur Verwendung einiger spezieller Parameter.

- PID Derivative Time: Einheiten sind in Zyklen definiert. Dies ist die Udaterate der Achse wie mit Set/GetSampleTime definiert. Z.B. falls der Wert auf 10 gesetzt wurde, wird der Derivative Term alle 10 Zyklen neu berechnet. PID Integrator Limit dagegen hat count*cycles und ein Scaling in Units. Dies gleicht dem Scaling und den Units der Position Loop Integrator Summe. Z.B. ein konstanter Schleppabstandsfehler von 100 Inkrementen der für die Dauer von 256 Zyklen auftritt, ergibt eine Inetrgatorsumme von 100*256=25600
- PID Integrator Gain: Das Scaling beträgt 1/256. Die Einstellung von 256 korrespondiert mit "Unity" vom Integrator Gain. Bezogen auf das obige Beispiel würde dies die Integrator Summe von 25600 veranlassen einen Beitrag zur PID Ausgabe von 25600 zu machen.
- PID Output Gain: ist ein Skalierungsfaktor zur Ausgabe des digitalen Servofilters in Prozenten der Ausgabe. Der Standardwert ist 65535 bzw. 100%. Um eine Ausgabe von 50% zu erreichen, muss der Wert auf 32767 eingestellt werden.
- Die BiQuad Koeffizienten konfigurieren die beiden Biquad Filter. Falls beide aktiviert wurden, wird die Ausgabe verkettet (Filter1 gefolgt von Filter2). Falls für eine Achse der Filter1 deaktiviert wurde, dann ist auch der ilter2 deaktiviert, egal welcher Wert übergeben wurde. Die Vorzeichen-relevanten Koeffizienten und der vorzeichenlose skalarer Wert K werden kombiniert zu folgender Gleichung für jeden Filter:
  $Y_n = K \times (B_0 \times X + B_1 \times X_{n-1} + B_2 \times X_{n-2} + A_1 \times Y_{n-1} * A_2 \times Y_{n-2})$
  Wobei Y_n ist die Filterausgabe in Zyklus n und X_n ist die Filtereingabe in Zyklus n.

Einschränkung

Set/GetPositionLoop ist ein gebuffertes Kommando. Alle Parameter die gesetzt wurden sind gebuffert und werden solange nicht in die aktiven Register übernommen bis ein Update-Kommando oder MultiUpdate-Kommando, oder Update aufgrund eines Breakpoints welches ausgelöst wurde, ausgeführt wurde. Die Werte die mit dem Kommando GetPositionLoop gelesen werden, sind die gebufferten Werte.

siehe

Update, Set/GetUpdateMask, MultiUpdate, Set/GetBreakPointUpdateMask, GetPositionLoopValue, Set/GetKi, Set/GetKd, Set/GetKp, Set/GetBiQuadCoefficient, Set/GetIntegrationLimit, Set/GetDerivativeTime, Set/GetKout, Set/GetKaff, Set/GetKvff

DLL: Delphi, C++, VB

wir verweisen auf die verschiedenen Importdateien (imp*.*)

MotionScript®

SetPositionLoop(hPosys#, axis, parameter, value)

Aufruf-konvention	GetPositionLoop(hPosys#, axis, parameter)

¤

Set/GetProfileKp

SetProfileKp
GetProfileKp

Syntax	SetProfileKp(hPosys, axis, Kp) stdcall GetProfileKp(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	Kp	unsigned 16 bits	Range	0 to 2^{15}-1
			Scaling	unity
Definition	Der "ProfileKp" ist ein proportional wirkender Verstärkerparameter mit einem Einstellbereich zwischen 0 und 32767. Wenn auf 32767 gesetzt wird keine gefilterte Ausgabe erzeugt und ist gleich dem Wert 0. Für Werte niedriger als 32767 vergrößert sich entsprechend der Filter. SetProfileKpsetzt die proportionale Beschleunigungsverstärkung des digitalen Servofilters für d spezifizierte Achse. Mit diesem Kommando ist es möglich im Trapez/Electronic Gearing und Geschwindigkeits-Modus S-Kurven ähnliche Profile zu erstellen mit dem Vorteil, dass alle Param (Position, Beschleunigung, Geschwindigkeit, assymetrische Profilerstellung etc.) wie im reinen Trapez-Modus jederzeit während des Verfahrens verändert werden können. GetProfileKpliest den programmierten Wert für die proportionale Beschleunigungsverstärkung die spezifizierte Achse.			
Einschränkung	SetProfileKp ist ein gebuffertes Kommando und wird erst nach einem Update bzw. MultiUpdate Kommando aktiviert. Dieses Kommando ist nur optional für POSYS® 800-B und 900 Serie v erfügbar. Für Details bitte kontaktieren Sie POSYS Motion Control GmbH & Co.KG			
siehe	Set/GetKd, Set/GetKi, Set/GetKout,Set/GetKaff, Set/GetKvff, MultiUpdate, Update			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetProfileKp(hPosys#, axis, Kp) GetProfileKp(hPosys#, axis)			

¤

Spezielle Piezo-Motorfunktionen

Set/GetpzIntLim

SetpzIntLim
GetpzIntLim

nur verfügbar für POSYS® 18xx-Piezo/19xx-Piezo Series

Syntax	SetpzIntLim(hPosys,axis, limit) stdcall GetpzIntLim(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
Ergebnis	Name	Type		
	limit	unsigned 32 bits	Range	0 to 2^{31}-1
			Scaling	$1/2^{8}$
			Units	counts*cycle
Definition	SetpzIntLimlädt den Integralanteil in den Servofilter für die spezifizierte Achse wenn Schleppabstandsfehler ist im primären Fenster. GetpzIntLimliest den programmierten Wert zurück wenn Schleppabstandsfehler ist im primäre Fenster. Rückgabewert = unsigned 32 Bits. Skalierungsbeispiel: Die Skalierung ist dieselbe wie für das Kommando Set/GetIntegrationLimit Ein konstanter Positionsfehler von 100 Zählern über eine Periode von 256 Zyklen würde einen Intergralwert von 100(100*256/256) zum Ergebnis haben, und daher würde ein IntegralLimit W von 100 den totalen, akkumulierten Integralfehler auf 25600 count*cycles begrenzen.			
Einschränkung	Dieses Kommando ist nicht verfügbar für die POSYS® 700/800/800-B/900/1800/1900 Serien. E ist nur gültig mit den POSYS® 1800-Piezo/1900-Piezo Serien.			
siehe	Set/GetpzKaff, Set/GetpzKp, Set/GetpzKi, Set/GetpzKd, Set/GetpzKvff, Set/GetpzKdPeriod, Set/GetpzWindow2, Set/GetpzKvff2			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetpzIntLim(hPosys#,axis, limit) GetpzIntLim(hPosys#, axis)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

SetpzIntLimitStl
GetpzIntLimitStl

nur verfügbar für POSYS® 18xx-Piezo/19xx-Piezo Series

Syntax	SetpzIntLimitStl(hPosys,axis, limit) stdcall GetpzIntLimitStl(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
Ergebnis	Name	Type		
	limit	unsigned 32 bits	Range	0 to 2^{31}-1
			Scaling	$1/2^{8}$
			Units	counts*cycle
Definition	SetpzIntLimitStlädt den Integralanteil in den Servofilter für die spezifizierte Achse wenn die Achse sich im programmierten SettleWindow befindet. GetpzIntLimitStlliest den programmierten Wert zurück. Rückgabewert = unsigned 32 Bits. Skalierungsbeispiel: Die Skalierung ist dieselbe wie für das Kommando Set/GetIntegrationLimit Ein konstanter Positionsfehler von 100 Zählern über eine Periode von 256 Zyklen würde einen Intergralwert von 100(100*256/256) zum Ergebnis haben, und daher würde ein IntegralLimit W von 100 den totalen, akkumulierten Integralfehler auf 25600 count*cycles begrenzen.			
Einschränkung	Dieses Kommando ist nicht verfügbar für die POSYS® 700/800/800-B/900/1800/1900 Serien. E ist nur gültig mit den POSYS® 1800-Piezo/1900-Piezo Serien.			
siehe	Set/GetpzOffsetNeg, Set/GetpzOffsetPos, Set/GetpzWindow			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetpzIntLimitstl(hPosys#,axis, limit) GetpzIntLimitStl(hPosys#, axis)			

¤

Set/GetpzKaff

SetpzKaff
GetpzKaff

nur verfügbar für POSYS® 18xx-Piezo/19xx-Piezo Series

Syntax	SetpzKaff(hPosys, axis, Kaff) stdcall GetpzKaff(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	Kaff	unsigned 16 bits	Range	0 to 2^{15}-1
			Scaling	unity
Definition	SetpzKaffsetzt die Acceleration Feedforward Verstärkung des digitalen Servofilters für die spezifizierte Achse wenn Schleppabstandsfehler ist im primären Fenster. GetpzKaffliest den programmierten Wert für die Acceleration Feedforward Verstärkung wenn Schleppabstandsfehler ist im primären Fenster. Rückgabewert = unsigned 16 Bits.			
Einschränkung	Dieses Kommando ist nicht verfügbar für die POSYS® 700/800/800-B/900/1800/1900 Serien. E ist nur gültig mit den POSYS® 1800-Piezo/1900-Piezo Serien.			
siehe	Set/GetpzKp, Set/GetpzKi, Set/GetpzKd, Set/GetpzKvff, Set/GetpzIntLim, Set/GetpzKdPeriod, Set/GetpzWindow2, Set/GetpzKvff2			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetpzKaff(hPosys#, axis, Kaff) GetpzKaff(hPosys#, axis)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

SetpzKd
GetpzKd

nur verfügbar für POSYS® 18xx-Piezo/19xx-Piezo Series

Syntax	SetpzKd(hPosys, axis, Kd) stdcall GetpzKd(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	Kd	unsigned 16 bits	Range	0 to 2^{15}-1
			Scaling	unity
Definition	SetpzKd setzt den differenzialen Verstärkeranteil für den digitalen Servofilter für die spezifiziert Achse wenn Schleppabstandsfehler ist im primären Fenster. GetpzKd liest den programmierten Wert des differentziellen Verstärkeranteils wenn Schleppabstandsfehler ist im primären Fenster. Rückgabewert = unsigned 16 Bits.			
Einschränkung	Dieses Kommando ist nicht verfügbar für die POSYS® 700/800/800-B/900/1800/1900 Serien. E ist nur gültig mit den POSYS® 1800-Piezo/1900-Piezo Serien.			
siehe	Set/GetpzKaff, Set/GetpzKp, Set/GetpzKi, Set/GetpzKvff, Set/GetpzIntLim, Set/GetpzKdPeriod, Set/GetpzWindow2, Set/GetpzKvff2			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetpzKd(hPosys#, axis, Kd) GetpzKd(hPosys#, axis)			

¤

Set/GetpzKdPeriod

SetpzKdPeriod
GetpzKdPeriod

nur verfügbar für POSYS® 18xx-Piezo/19xx-Piezo Series

Syntax	SetpzKdPeriod(hPosys, axis, time) stdcall GetpzKdPeriod(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	time	unsigned 16 bits	Range	0 to 2^{15}-1
			Scaling	unity
			Units	cycles
Definition	SetpzKdPeriodsetzt den Multiplikator in Chipzyklen des Servofilters zur Kalkulation des Differenzialanteils für die spezifizierte Achse. GetpzKdPeriodliest den programmierten Wert zurück. Rückgabewert = unsigned 16 Bits.			
Einschränkung	Dieses Kommando ist nicht verfügbar für die POSYS® 700/800/800-B/900/1800/1900 Serien. E ist nur gültig mit den POSYS® 1800-Piezo/1900-Piezo Serien.			
siehe	Set/GetpzKaff, Set/GetpzKp, Set/GetpzKi, Set/GetpzKd, Set/GetpzKvff, Set/GetpzIntLim, Set/GetpzWindow2, Set/GetpzKvff2			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetpzKdPeriod(hPosys#, axis, time) GetpzKdPeriod(hPosys#, axis)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

SetpzKi
GetpzKi

nur verfügbar für POSYS® 18xx-Piezo/19xx-Piezo Series

Syntax	SetpzKi(hPosys, axis, Ki) stdcall GetpzKi(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	Ki	unsigned 16 bits	Range	0 to 2^{15}-1
			Scaling	unity
Definition	SetpzKi setzt die Integralverstärkung für den digitalen Servofilter für die spezifizierte Achse we Schleppabstandsfehler ist im primären Fenster. GetpzKi liest den programmierten Wert für die Integralverstärkung des digitalen Servofilters fü die spezifizierte Achse wenn Schleppabstandsfehler ist im primären Fenster. Rückgabewert = unsigned 16 Bits.			
Einschränkung	Dieses Kommando ist nicht verfügbar für die POSYS® 700/800/800-B/900/1800/1900 Serien. E ist nur gültig mit den POSYS® 1800-Piezo/1900-Piezo Serien.			
siehe	Set/GetpzKaff, Set/GetpzKp, Set/GetpzKd, Set/GetpzKvff, Set/GetpzIntLim, Set/GetpzKdPeriod, Set/GetpzWindow2, Set/GetpzKvff2			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetpzKi(hPosys#, axis, Ki) GetpzKi(hPosys#, axis)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

SetpzKp
GetpzKp

nur verfügbar für POSYS® 18xx-Piezo/19xx-Piezo Series

Syntax	SetpzKp(hPosys, axis, Kp) stdcall GetpzKp(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente Ergebnis	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	Kp	unsigned 16 bits	Range	0 to 2^{15}-1
			Scaling	unity
Definition	SetpzKp setzt die proportionale Verstärkung des digitalen Servofilters für die spezifizierte Achse wenn Schleppabstandsfehler ist im primären Fenster. GetpzKp liest den programmierten Wert für die proportionale Verstärkung des digitalen Servofilters für die spezifizierte Achse wenn Schleppabstandsfehler ist im primären Fenster. Rückgabewert = unsigned 16 Bits.			
Einschränkung	Dieses Kommando ist nicht verfügbar für die POSYS® 700/800/800-B/900/1800/1900 Serien. E ist nur gültig mit den POSYS® 1800-Piezo/1900-Piezo Serien.			
siehe	Set/GetpzKaff, Set/GetpzKi, Set/GetpzKd, Set/GetpzKvff, Set/GetpzIntLim, Set/GetpzKdPeriod, Set/GetpzWindow2, Set/GetpzKvff2			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetKp(hPosys#, axis, Kp) GetKp(hPosys#, axis)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

SetpzKvff
GetpzKvff

nur verfügbar für POSYS® 18xx-Piezo/19xx-Piezo Series

Syntax	SetpzKvff(hPosys, axis, Kvff) stdcall GetpzKvff(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	Kvff	unsigned 16 bits	Range	0 to 2^{15}-1
			Scaling	unity
Definition	SetpzKvffsetzt die Velocity Feedforward Verstärkung des digitalen Servofilters für die spezifizie Achse wenn Schleppabstandsfehler ist im primären Fenster. GetpzKvffliest den programmierten Wert für die Velocity Feedforward Verstärkung wenn Schleppabstandsfehler ist im primären Fenster. Rückgabewert = unsigned 16 Bits.			
Einschränkung	Dieses Kommando ist nicht verfügbar für die POSYS® 700/800/800-B/900/1800/1900 Serien. E ist nur gültig mit den POSYS® 1800-Piezo/1900-Piezo Serien.			
siehe	Set/GetpzKaff, Set/GetpzKp, Set/GetpzKi, Set/GetpzKd, Set/GetpzIntLim, Set/GetpzKdPeriod, Set/GetpzWindow2, Set/GetpzKvff2			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetpzKvff(hPosys#, axis, Kvff) GetpzKvff(hPosys#, axis)			

¤

Set/GetpzKvff2

SetpzKvff2
GetpzKvff2

nur verfügbar für POSYS® 18xx-Piezo/19xx-Piezo Series

Syntax	SetpzKvff2(hPosys, axis, Kvff) stdcall GetpzKvff2(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	Kvff	unsigned 16 bits	Range	0 to 2^{15}-1
			Scaling	unity
Definition	SetpzKvff2setzt die Velocity Feedforward Verstärkung des digitalen Servofilters für die spezifizierte Achse wenn Schleppabstandsfehler ist im sekundären Fenster. GetpzKvff2liest den programmierten Wert für die Velocity Feedforward Verstärkung wenn Schleppabstandsfehler ist im sekundären Fenster. Rückgabewert = unsigned 16 Bits.			
Einschränkung	Dieses Kommando ist nicht verfügbar für die POSYS® 700/800/800-B/900/1800/1900 Serien. E ist nur gültig mit den POSYS® 1800-Piezo/1900-Piezo Serien.			
siehe	Set/GetpzKaff, Set/GetpzKp, Set/GetpzKi, Set/GetpzKd, Set/GetpzKvff, Set/GetpzIntLim, Set/GetpzKdPeriod, Set/GetpzWindow2			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetpzKvff2(hPosys#, axis, Kvff) GetpzKvff2(hPosys#, axis)			

¤

Set/GetpzOffsetNeg

SetpzOffsetNeg
GetpzOffsetNeg

nur verfügbar für POSYS® 18xx-Piezo/19xx-Piezo Series

Syntax	SetpzOffsetNeg(hPosys, axis, offset) stdcall GetpzOffsetNeg(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	offset	unsigned 16 bit	Range	0 to 2^{15}-1
			Scaling	unity
Definition	SetpzOffsetNegist ein Servovorgabewert der verwendet wird wenn der Motor nicht in-Position ist, definiert mit SetSettleWindow und SetSettleTime (GetActivityStatus, Bit=7) und der Schleppabstandsfehler ist negativ. GetpzOffsetNegliefert den aktuellen Wert für die gewählte Achse zurück. Rückgabewert = unsigned 16 Bits.			
Einschränkung	Dieses Kommando ist nicht verfügbar für die POSYS® 700/800/800-B/900/1800/1900 Serien. E ist nur gültig mit den POSYS® 1800-Piezo/1900-Piezo Serien.			
siehe	Set/GetpzOffsetPos, Set/GetpzIntLimStl, Set/GetpzWindow			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetpzOffsetNeg(hPosys#, axis, offset) GetpzOffsetNeg(hPosys#, axis)			

¤

Set/GetpzOffsetPos

SetpzOffsetPos
GetpzOffsetPos

nur verfügbar für POSYS® 18xx-Piezo/19xx-Piezo Series

Syntax	SetpzOffsetPos(hPosys, axis, offset) stdcall GetpzOffsetPos(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	offset	unsigned 16 bit	Range	0 to 2^{15}-1

	Scaling unity
Definition	SetpzOffsetPosist ein Servovorgabewert der verwendet wird wenn der Motor ist nicht in-Positi ist, definiert mit SetSettleWindow und SetSettleTime (GetActivityStatus, Bit=7) und der Schleppabstandsfehler ist positiv. GetpzOffsetPosliefert den aktuellen Wert für die gewählte Achse zurück. Rückgabewert = unsigned 16 Bits.
Einschränkung	Dieses Kommando ist nicht verfügbar für die POSYS® 700/800/800-B/900/1800/1900 Serien. E ist nur gültig mit den POSYS® 1800-Piezo/1900-Piezo Serien.
siehe	Set/GetpzOffsetNeg, Set/GetpzIntLimStl, Set/GetpzWindow
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)
MotionScript® Aufruf-konvention	SetpzOffsetPos(hPosys#, axis, offset) GetpzOffsetPos(hPosys#, axis)

¤

Set/GetpzWindow

SetpzWindow
GetpzWindow

nur verfügbar für POSYS® 18xx-Piezo/19xx-Piezo Series

Syntax	SetpzWindow(hPosys, axis, window) stdcall GetpzWindow(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	window	unsigned 16 bits	Range	0 to 2^{16}-1
			Scaling	unity
			Units	counts
Definition	SetpzWindow definiert die Fenstergröße innerhalb der sich eine spezifizierte Achse befinden mu um die anderen Piezo-spezifischen Kommandos zu aktivieren. GetpzWindow liest den programmierten Wert zurück. Rückgabewert = unsigned 16 Bits.			
Einschränkung	Dieses Kommando ist nicht verfügbar für die POSYS® 700/800/800-B/900/1800/1900 Serien. E ist nur gültig mit den POSYS® 1800-Piezo/1900-Piezo Serien.			
siehe	Set/GetpzOffsetNeg, Set/GetpzOffsetPos, Set/GetpzIntLimStl			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetpzWindow(hPosys#, axis, window) GetpzWindow(hPosys#, axis)			

¤

Set/GetpzWindow2

SetpzWindow2
GetpzWindow2

nur verfügbar für POSYS® 18xx-Piezo/19xx-Piezo Series

Syntax	SetpzWindow2(hPosys, axis, window) stdcall GetpzWindow2(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	window	unsigned 16 bits	Range	0 to 2^{16}-1
			Scaling	unity
			Units	counts
Definition	SetpzWindow2definiert die Fenstergröße für das zweite Fenster innerhalb der sich eine spezifizierte Achse befinden muss um die anderen Piezo-spezifischen Kommandos zu aktivieren GetpzWindow2liest den programmierten Wert zurück. Rückgabewert = unsigned 16 Bits.			
Einschränkung	Dieses Kommando ist nicht verfügbar für die POSYS® 700/800/800-B/900/1800/1900 Serien. E ist nur gültig mit den POSYS® 1800-Piezo/1900-Piezo Serien.			
siehe	Set/GetpzKaff, Set/GetpzKp, Set/GetpzKi, Set/GetpzKd, Set/GetpzKvff, Set/GetpzIntLim, Set/GetpzKdPeriod, Set/GetpzKvff2			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetpzWindow2(hPosys#, axis, window) GetpzWindow2(hPosys#, axis)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

Enkoder

AdjustActualPosition

AdjustActualPosition

Syntax	AdjustActualPosition(hPosys, axis, position) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	position	signed 32 bits	Range	-2^{31} to $2^{31}-1$
			Scaling	unity
			Units	counts steps
Definition	Die als der Parameter für AdjustActualPosition angegebene Position wird addiert mit dem aktuellen Positionsregister (Enkoderposition) für die spezifizierte Achse. Dies hat den Effekt, das ein Offset addiert bzw. subtrahiert wird von der aktuellen Position. Zur selben Zeit wird die gegenwärtige Sollposition ersetzt mit der Istposition abzgl. dem gegenwärtigen Istpositionsfehle Dies verhindert einen abrupten Sprung wenn die neue Position aktuell wird. Die Zielposition (siehe SetPosition) wird auch mit diesem Wert modifiziert, sodass keine Bewegu stattfindet wenn ein Update-Kommando initiiert wird. In Wirklichkeit etabliert diese Anweisung einen neuen Bezug von wo aus neue Positionen kalkuliert werden. Gewöhnlich wird es eingesetz um nach einer Referenzfahrt die Position neu zu definieren. Anmerkung: Bei den Achsen in Schrittmotormodus wird der Istpositionsfehler genullt. AdjustActualPosition wirkt sofort, es ist nicht gebuffert und benötigt kein zusätzliches Update-Kommando.			
Einschränkung				
siehe	GetPositionError, GetActualVelocity, Set/GetActualPositionUnits, Set/GetActualPosition			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	AdjustActualPosition(hPosys#,axis,position)			

¤

GetAbsoluteSSIPosition

Syntax	GetAbsoluteSSIPosition(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
Ergebnis	Liest die Position eines Absolutwertgebers als 32-Bit Integerwert.			
Definition	GetAbsoluteSSIPosition liefert ein 32-Bit Integerwert von der Position des Absolutwertgebers für die spezifizierte Achse.			
Einschränkung	Diese Funktion ist nur in Verbindung mit dem SSI800/900 (Synchronous Serial Interface) verwendbar.			
siehe	GetSSIVersion, Set/GetEncoderSource, ResetSSI, Read/WriteSSIResolution, Read/WriteSSIRegister, Read/WriteSSIFrequency, SetActualToAbsolutePosition			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	GetAbsoluteSSIPosition(hPosys#, axis)			

¤

GetActualVelocity

Syntax	GetActualVelocity(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
Ergebnis	Name	Type		
	velocity	signed 32 bits	Range	-2^{31} to $2^{31}-1$
			Scaling	$1/2^{16}$
			Units	counts/cycle
Definition	GetActualVelocity liest die aktuelle Geschwindigkeit für die spezifizierte Achse. Der Wert ist das Ergebnis des letzten Enkodereingangs, daher wird es akkurat sein innerhalb einer Updaterate. Skalierungsbeispiel: Falls ein Wert von 1234567 vom GetActualVelocity-Kommando empfangen wurde (high-Word - 0FFEDh, low-Word - 2979h), entspricht dies einer Geschwindigkeit von 1234567/65536 oder 18,8380 counts/cycle.			
Einschränkung	Die aktuelle Geschwindigkeit wird ermittelt indem die aktuelle Position für den vorhergehenden Arbeitszyklus von der aktuellen Position des gegenwärtigen Arbeitzyklus subtrahiert wird. Das Ergebnis der Subtraktion wird immer ein Integer sein, da die Positionsangabe immer ein Integer ist. Als ein Ergebnis wird der Wert der zurückgeliefert wird durch das Kommando GetActualVelocity immer ein Vielfaches von 65536 sein, da dies ein Wert repräsentiert im Format 16.16. Das niederwertige Wort ist immer Null.			
siehe	GetCommandedVelocity			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	GetActualVelocity(hPosys#, axis)			

¤

GetCaptureValue

Syntax	GetCaptureValue(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
Ergebnis		Type		
	captured position	signed 32 bits	Range	-2^{31} to $2^{31}-1$
			Scaling	unity
			Units	counts steps
Definition	GetCaptureValueliefert den Inhalt des Position C apture Registers für die spezifizierte Achse. Dieses Kommando setzt auch die Erfassungshardware (Trace) zurück um eine weitere Erfassun erlauben. Falls Set/GetActualPositionUnits auf “steps” eingestellt ist, ist der zurückgelieferte Wert in Einhei von Schritten.			
Einschränkung				
siehe	Set/GetCaptureSource, Set/GetActualPositionUnits, GetActivityStatus			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	GetCaptureValue(hPosys#, axis)			

¤

GetSSIVersion

Syntax	GetSSIVersion(hPosys) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
Ergebnis	Name	Type		
	version	unsigned 16 bit		
Definition	Diese Funktion liefert die Versionsnummer des optional installierten SSI-Boards zurück. Es liefert nützliche Informationen bei der Problemanalyse.			
Einschränkung	Diese Funktion ist nur in Verbindung mit dem SSI800/900 (Synchronous Serial Interface) verwendbar.			
siehe	GetAbsoluteSSIPosition, Set/GetEncoderSource, ResetSSI, Read/WriteSSIResolution, Read/WriteSSIRegister, Read/WriteSSIFrequency, SetActualToAbsolutePosition			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	GetSSIVersion(hPosys#)			

¤

Read/WriteSSIFrequency

WriteSSIFrequency
ReadSSIFrequency

Syntax	WriteSSIFrequency(hPosys, axis, resolution) stdcall ReadSSIFrequency(hPosys, axis, resolution) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	frequency	1100 KHz 550 KHz 275 KHz 137.5 KHz	1100 550 275 137	
Definition	WriteSSIFrequency überschreibt die Standardeinstellungen für die Frequenz (1,1 MHz Frequen Jede Achse kann auf eigene Werte gesetzt werden. ReadSSIFrequency liest die programmierten Werte für die spezifizierte Achse.			
Einschränkung	Verändern Sie keinesfalls die F requenz während die Achse in Bewegung ist und die Verstärker eingeschaltet sind! Diese Funktion ist nur in Verbindung mit der POSYS® 800 (A und B Serie) und 1800 (PC /104) der POSYS® 900 und 1900 (PC I) verfügbar mit installiertem SSI800/900.			
siehe	GetSSIVersion, GetAbsoluteSSIPosition, Set/GetEncoderSource, ResetSSI, Read/WriteSSIResolution, Read/WriteSSIRegister, SetActualToAbsolutePosition			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	WriteSSIFrequency(hPosys#, axis, resolution) ReadSSIFrequency(hPosys#, axis, resolution)			

¤

Read/WriteSSIRegister

WriteSSIRegister
ReadSSIRegister

Syntax	WriteSSIRegister(hPosys, axis, resolution, frequency) stdcall ReadSSIRegister(hPosys, axis, resolution, frequency) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	resolution	25 bits 13 bits 12 bits 10 bits	25 13 12 10	
	frequency	1.1 MHz 550 KHz 275 KHz 137.5 KHz	1100 550 275 137	
Definition	WriteSSIRegisterüberschreibt die Standardeinstellungen für die Frequenz (1,1 MHz Frequenz) und die DIP-Schaltereinstellungen für die Auflösung. Jede Achse kann auf eigene Werte gesetzt werden. ReadSSIRegisterliest die programmierten Werte für die spezifizierte Achse.			
Einschränkung	Verändern Sie keinesfalls die Auflösung während die Achse in Bewegung ist und die Verstärker eingeschaltet sind! Diese Funktion ist nur in Verbindung mit der POSYS® 800 (A und B Serie) und 1800 (PC /104) der POSYS® 900 und 1900 (PC I) verfügbar mit installiertem SSI800/900.			
siehe	GetSSIVersion, GetAbsoluteSSIPosition, Set/GetEncoderSource, ResetSSI, Read/WriteSSIResolution, Read/WriteSSIFrequency, SetActualToAbsolutePosition			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	WriteSSIRegister(hPosys#, axis, resolution, frequency) ReadSSIRegister(hPosys#, axis, resolution, frequency)			

¤

Read/WriteSSIResolution

WriteSSIResolution
ReadSSIResolution

Syntax	WriteSSIResolution(hPosys, axis, resolution) stdcall ReadSSIResolution(hPosys, axis, resolution) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	resolution	25 bits 13 bits 12 bits 10 bits	25 13 12 10	
Definition	WriteSSIResolutionüberschreibt die Standardeinstellungen für die Auflösung (egal wie die DIP-Schalter stehen). Jede Achse kann auf eigene Werte gesetzt werden. ReadSSIResolutionliest die programmierten Werte für die spezifizierte Achse.			
Einschränkung	Verändern Sie keinesfalls die Auflösung während die Achse in Bewegung ist und die Verstärker eingeschaltet sind! Diese Funktion ist nur in Verbindung mit der POSYS® 800 (A und B Serie) und 1800 (PC /104) der POSYS® 900 und 1900 (PC I) verfügbar mit installiertem SSI800/900.			
siehe	GetSSIVersion, GetAbsoluteSSIPosition, Set/GetEncoderSource, ResetSSI, Read/WriteSSIRegister, Read/WriteSSIFrequency, SetActualToAbsolutePosition			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	WriteSSIResolution(hPosys#, axis, resolution) ReadSSIResolution(hPosys#, axis, resolution)			

¤

ResetSSI

Syntax	ResetSSI(hPosys) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
Definition	ResetSSI setzt das SSI 800/900 Board zurück auf die Standardwerte. Frequenz = 1,1 MHz und eine Resolution entsprechend den DIP-Schaltereinstellungen. Nach einem Reset wird die aktuelle absolute Position des Enkoders an die POSYS® geschickt. Für weitere Informationen verweisen wir auf das Handbuch zum SSI800/900.			
Einschränkung	Ein externer Reset bzw. ein Reset ausgelöst mit dem Kommando HardReset oder SoftReset resettet nicht die SSI 800/900. Nur das Ausschalten bzw. das Kommando ResetSSI verursacht ein Reset der SSI 800/900.			
siehe	GetSSIVersion, GetAbsoluteSSIPosition, Set/GetEncoderSource, Read/WriteSSIResolution, Read/WriteSSIRegister, Read/WriteSSIFrequency, SetActualToAbsolutePosition			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	ResetSSI(hPosys#)			

¤

SetActualToAbsolutePosition

Syntax	SetActualToAbsolutePosition(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
Definition	Diese Funktion setzt die aktuelle Position der POSYS® auf die absolute Enkoderposition für die spezifizierte Achse. Es synchronisiert 2 Zähler. Die POSYS® bewahrt den vollen 32-Bit Positionsbereich, die SSI 800/900 bewahrt die Position des Absolutwertgeberbereiches. Wenn die Auflösung des Enkoders 25-Bit beträgt, wird die Positionsangabe 0-33554432 sein und wieder bei 0 starten.			
Einschränkung	Mann muss Vorsicht walten lassen bei der Verwendung dieser Funktion um sicher zu stellen, das keine abrupte Bewegung stattfindet. Ehe Motoren und Verstärker eingeschaltet werden, sollte sichergestellt sein, dass die Sollposition und die aktuelle Position nicht mehr als ein paar Zähler voneinander variieren, weil ansonsten plötzliche und gefährliche Bewegungen initiiert werden könnten, wenn das System anfängt zu regeln! SetMotorMode sollte auf OFF sein (nicht regeln) e das Kommando SetActualToAbsolutePosition ausgeführt wird. Anmerkung: Diese Funktion ist nur in Verbindung mit dem SSI800/900 (Sy nchrono Serial Interface) v erwendbar.			
siehe	GetSSIVersion, GetAbsoluteSSIPosition, Set/GetEncoderSource, ResetSSI, Read/WriteSSIResolution, Read/WriteSSIRegister, Read/WriteSSIFrequency, SetActualToAbsolutePosition, Set/GetActualPosition			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetActualToAbsolutePosition(hPosys#, axis)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

SetActualPosition
GetActualPosition

Syntax	SetActualPosition(hPosys, axis, position) stdcall GetActualPosition(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	position	signed 32 bit	Range	-2^{31} to $2^{31}-1$
			Scaling	unity
			Units	counts steps
Definition	SetActualPositionlädt die aktuelle Positionsinformation (Enkoderposition in den Positionsregist der spezifizierten Achse. Zur gleichen Zeit wird die Sollposition mit dem neuen, geladenen Wert abzgl. dem aktuellen Schleppabstandsfehler ersetzt. Dies vermeidet einen Sprung der Achse, w der neue Positionswert übernommen wird. Die Zielposition (SetPosition Kommando) wird ebens um diesen Wert aktualisiert, sodass keine Bahnbewegung stattfindet, wenn das Update Kommar ausgeführt wird. Dieses Kommando hat sogar Effekt auf die Kalkulation aller nachfolgenden Positionsangaben. Es wird im Allgemeinen verwendet nach dem Anfahren eines Referenzpunkte um die Position neu zu definieren. SetActualPositionwird nicht gebuffert, sondern wird sofort aktualisiert. GetActualPositionliest den Inhalt des Registers für die Istposition. Der Inhalt dieses Registers der Wert des letzten Enkodereingangssignales, welches somit eine Genauigkeit innerhalb eines Zyklus aufweist (wie mit dem Kommando Set/GetSampleTime definiert).			
Einschränkung	Anmerkung: Bei den Schrittmotorversionen wird die aktuelle Position in Schritte definiert. Dies stellt sicher, dass sowohl für den Bahngenerator und für die aktuelle Position gleiche Einheiten verwendet werden. Für gewöhnlich hat der Bahngenerator (Regler) eine höhere Auflösung als d Enkoder, somit ist maximale Genauigkeit gewährleistet bei der Verwendung dieser Methode. Anmerkung: Bei den Schrittmotorversionen wird der aktuelle Schleppabstandsfehler genullt.			
siehe	GetPositionError, GetActualVelocity, Set/GetActualPositionUnits, AdjustActualPosition			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetActualPosition(hPosys#, axis, position) GetActualPosition(hPosys#, axis)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

SetActualPositionUnits
GetActualPositionUnits

Syntax	SetActualPositionUnits(hPosys, axis, mode) stdcall GetActualPositionUnits(hPosys, axis) stdcall			
Motortyp			Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	mode	counts steps	0 1	
Definition	SetActualPositionUnits bestimmt die Einheiten von GetActualPosition wenn es die Position der spezifizierten Achse zurückliefert. Wenn auf C ounts eingestellt, liefert GetActualPosition die Achsenposition in Enkodereinheiten zurück. Wenn auf Steps eingestellt, dann liefert GetActualPosition die Achsenposition in Schritten (Takte) zurück. GetActualPositionUnits liefert den programmierten Modus zurück.			
Einschränkung				
siehe	Set/GetActualPosition, Set/GetEncoderToStepRatio, AdjustActualPosition, GetCaptureValue			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetActualPositionUnits(hPosys#, axis, mode) GetActualPositionUnits(hPosys#, axis)			

¤

Set/GetCaptureSource

SetCaptureSource
GetCaptureSource

Syntax	SetCaptureSource(hPosys, axis, source) stdcall GetCaptureSource(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	source	index home	0 1	
Definition	SetCaptureSource bestimmt welche der beiden Enkodersignale, Index oder Home, verwendet wird die High-Speed Erfassung für die aktuelle Achsenposition für die spezifizierte Achse, zu verwenden. GetCaptureSource liefert den programmierten Zustand zurück.			
Einschränkung				
siehe	GetCaptureValue			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetCaptureSource(hPosys#, axis, source) GetCaptureSource(hPosys#, axis)			

¤

Set/GetEncoderModulus

SetEncoderModulus
GetEncoderModulus

Syntax	SetEncoderModulus(hPosys, axis, modulus) stdcall GetEncoderModulus(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	modulus	unsigned 16 bits	Range	1 to 2^{16}-1
			Scaling	unity
			Units	counts
Definition	SetEncoderModulus setzt den Parallelwortbereich für die spezifizierte Achse wenn Parallelwortrückmeldeeinheiten (Resolver, Absolutenkoder etc.) verwendet werden. Modulus bestimmt den Bereich für das angeschlossene Gerät. Der Wert sollte die Hälfte des aktuellen Modulus der Achse betragen. z.B. wenn in Verbindung mit einem linearen Potentiometer, verbunden mit einem AD-Wandler, d Parallelworteingang mit 12-Bit Auflösung verwendet wird, dann ist der gesamte Bereich 4096 ur ein Wert von 2048 muss mit dem Kommando geladen werden. GetEncoderModulus liest den programmierten Wert zurück.			
Einschränkung	Dieses Kommando wird nur in Verbindung mit Parallelwortrückmeldeeinheiten (Option) verwendet. Falls inkrementale Enkoder verwendet werden, ist dieses Kommando nicht notwendig.			
siehe	Set/GetEncoderSource			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetEncoderModulus(hPosys#, axis, modulus) GetEncoderModulus(hPosys#, axis)			

¤

Set/GetEncoderSource

SetEncoderSource
GetEncoderSource

Syntax	SetEncoderSource(hPosys, axis, source) stdcall GetEncoderSource(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	source	incremental parallel none loopback (nur POSYS® 1800/1900 Serie)	0 1 2 3	
Definition	SetEncoderSource setzt die Art von Enkoder (inkremental oder Parallelwort (option)) für die spezifizierte Achse. Falls inkrementale Enkoder ausgewählt wurde, akzeptiert die POSYS® A un Signale und optionale Indexsignale. Wenn Parallelwort ausgewählt wurde, akzeptiert die POSYS entweder die SSI 800/900 oder eine andere, kundenspezifische Art der Parallelwortrückmeldun mit 16 Bit Wort Auflösung, die die aktuelle Positionsangabe für jede Achse beinhaltet. Externe Rückmeldeeinheiten mit weniger als 16-Bit können verwendet werden, müssen aber mit Nullen aufgefüllt werden. Wenn der Motortyp Pulse & Direction ist und der EncoderSource ist eingestellt au/f Loopback (n POSYS® 1800/1900 Serie), wird die Taktausgabe intern in den Zähler zurückgeführt. Dies erlau Positionserfassung für den Taktausgang falls ein Enkoder physikalisch nicht angeschlossen ist. Ansonsten ist es auch möglich das Kommando GetCommandedPosition zu verwenden um die Sollposition zu definieren und eine Positionserfassung aufgrund der ausgegebenen Takte zu bestimmen (z.B. bei POSYS® 85x-B und 95x). Bei beiden Modi, Loopback als auch GetCommandedPosition, wird keine aktuelle Positionsrückmeldung ermöglicht, sondern nur die aufgrund der Taktausgabe definierte Sollposition. GetEncoderSource liest den Code zurück für die eingestellte Enkoderart.			
Einschränkung				
siehe	Set/GetEncoderModulus			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetEncoderSource(hPosys#, axis, source) GetEncoderSource(hPosys#, axis)			

¤

Set/GetEncoderToStepRatio

SetEncoderToStepRatio
GetEncoderToStepRatio

Syntax	SetEncoderToStepRatio(hPosys, axis, counts, steps) stdcall GetEncoderToStepRatio(hPosys, axis) stdcall			
Motortyp			Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	counts	unsigned 16 bits	Range	0 to 2^{15}-1
			Scaling	unity
			Units	counts
	steps	unsigned 16 bits	Range	0 to 2^{15}-1
			Scaling	unity
			Units	steps
Definition	SetEncoderToStepRatio	setzt das Verhältnis von Encodersignalen zur Anzahl Schritte pro Motorumdrehung um die Anzahl erfasste Enkodersignale in Schritte zu konvertieren. "C ounts" die Enkoderauflösung pro Motorumdrehung. "Steps" ist die Anzahl Schritte pro Motorumdrehung Da dieses Kommando ein Verhältnis definiert, müssen die Eingaben nicht eine volle Umdrehung repräsentieren um korrekte Werte zurück zu liefern, sofern das Verhältnis stimmt.		
	GetEncoderToStepRatio	liest den programmierten Wert zurück.		
Einschränkung				
siehe	Set/GetActualPositionUnits			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetEncoderToStepRatio(hPosys#, axis, counts, steps) GetEncoderToStepRatio(hPosys#, axis)			

¤

On-Board RAM

DisableDPRam

nur gültig für POSYS® 18xx mit Dual Port RAM

Syntax	DisableDPRam(hPosys) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
Ergebnis	Deaktiviert den Zugriff auf das Dual Port RAM über den ISA-Bus.			
Definition	DisableDPRamdeaktiviert den DPRAM Modus auf der POSYS® 1800. Der On-Board Speicher is dann nur noch verwendbar im Single Port RAM Modus was langsamere Schreib- und Lesevorgär erlaubt. Dieses Kommando soll helfen auf PC -Systemen die ansonsten Probleme bereiten könn bei sofortiger Aktivierung des DPRAM, ein Aufhängen beim Bootvorgang zu vermeiden, egal ob Speicherbereich als frei gekennzeichnet ist oder nicht. Auch wenn beim späteren Betrieb eine sichere Verwendung des DPRAM über den ISA-Bus nicht gewährleistet ist, steht damit der On-Board Speicher zur weiteren Verwendung im SPRAM-Modus bereit. Mit dem Kommando EnableDPRam kann der Dual Port RAM Modus jederzeit wieder aktiviert werden.			
Einschränkung	Dieses Kommando ist nur für die POSYS® 1800 mit Dual Port RAM verfügbar. POSYS® Karten für den PCI-Bus mit Dual Port RAM sind hiervon nicht betroffen.			
siehe	EnableDPRam, ReadDPRamLong, WriteDPRamLong			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	DisableDPRam(hPosys#)			

¤

EnableDPRam

EnableDPRam

nur gültig für POSYS® 18xx mit Dual Port RAM

Syntax	EnableDPRam(hPosys) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
Ergebnis	Aktiviert den Zugriff auf das Dual Port RAM über den ISA-Bus.			
Definition	EnableDPRamaktiviert den DPRAM auf der POSYS® 1800. Der On-Board Speicher ist dann sowohl für den Zugriff über den ISA-Bus als auch über den Motion Prozessor freigeschaltet. Ohr diese Aktivierung ist der Speicher nur im Modus für Single Port RAM freigeschaltet. Dieses Kommando soll helfen auf PC -Systemen die ansonsten Probleme bereiten könnten bei sofortige Aktivierung des DPRAM, ein Aufhängen beim Bootvorgang zu vermeiden, egal ob der Speicherbereich als frei gekennzeichnet ist oder nicht. Mit den Kommandos ReadDPRamLong ur WriteDPRamLong kann der Speicher über den ISA-Bus direkt gelesen und beschrieben werden. Mit dem Kommando DisableDPRam kann der Dual Port RAM Modus jederzeit wieder deaktiviert werden.			
Einschränkung	Dieses Kommando ist nur für die POSYS® 1800 mit Dual Port RAM verfügbar. POSYS® Karten für den PCI-Bus mit Dual Port RAM sind hiervon nicht betroffen.			
siehe	DisableDPRam, ReadDPRamLong, WriteDPRamLong			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	EnableDPRam(hPosys#)			

¤

ReadBuffer

ReadBuffer

Syntax	ReadBuffer(hPosys, bufferID) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
		Type		
	bufferID	unsigned 16 Bit	Range	0 to 31
			Scaling	unity
			Units	-
Ergebnis		Type		
	data	signed 32 Bit	Range	-2^{31} to $2^{31}-1$
			Scaling	unity
			Units	-
Definition	ReadBuffer liefert einen 32-Bit Inhalt der aktuellen Position im spezifizierten Buffer. Die aktuelle Position wird bestimmt in dem die Basisadresse des Buffers (gesetzt durch Kommando SetBufferStart) dem Read Index des Buffers (gesetzt durch das Kommando SetBufferReadIndex) hinzuaddiert wird. Nachdem der Inhalt gelesen wurde, wird der ReadIndex um 1 inkrementiert; falls das Ergebnis gleich der Bufferlänge entspricht (gesetzt mit dem Kommando SetBufferLength),			

	wird der ReadIndex auf Null zurückgesetzt.
Einschränkung	Einige Kommandos ändern automatisch den ReadIndex, wie z.B. wenn ein Trace komplettiert wurde im “Rolling Mode”. Bereich für bufferID: 0 bis 31 Ehe der Buffer gelesen werden kann muss ReadIndex(n) aufgerufen werden n = Startposition des Buffers
siehe	Set/GetBufferReadIndex, WriteBuffer, Set/GetBufferStart, Set/GetBufferLength
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)
MotionScript® Aufruf-konvention	ReadBuffer(hPosys#, bufferID)

¤

ReadDPRamLong

nur gültig für POSYS® 18xx/19xx mit Dual Port RAM

Syntax	ReadDPRamLong(hPosys;offset) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	offset	unsigned 32 Bit	Speicherbasisadresse + offset	
Ergebnis		Type		
	data	signed 32 Bit	Range	-2^{31} to $2^{31}-1$
			Scaling	unity
			Units	-
Definition	Liest einen 32-Bit Wert von der Speicherbasisadresse + Offset			
Einschränkung	Dieses Kommando ist nur für POSYS® Karten mit Dual Port RAM verfügbar.			
siehe	WriteDPRamLong, EnableDPRam, DisableDPRam			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	ReadDPRamLong(hPosys#,offset)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

SetBufferFunction
GetBufferFunction

Syntax	SetBufferFunction(hPosys, axis, function, bufferID) stdcall GetBufferFunction(hPosys, axis, function) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	function	Position Velocity Acceleration Jerk Time AxisOut1	0 1 2 3 4 5 (nur verfügbar für POSYS 1800 & 1900 mit Firmware 5842018FD = Piezo Version)	
		Type	Range	
	bufferID	signed 16 bits	-1 to 31	
Definition	SetBufferFunction bestimmt die Interpretation der Daten, die im Buffer gespeichert werden, wenn eine Achse im External Profile Mode ist. Eine Funktion wird keinen assoziierten Buffer habe wenn der bufferID Parameter = -1 ist. Dies ist nützlich um eine Funktion zu deaktivieren. GetBufferFunction liefert die bufferID für die spezifizierte Funktion. Falls einer Funktion noch k Buffer zugewiesen wurde, ist der Wert -1.			
Einschränkung				
siehe	Set/GetProfileMode			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetBufferFunction(hPosys#, axis, function, bufferID) GetBufferFunction(hPosys#, axis, function)			

¤

Set/GetBufferLength

SetBufferLength
GetBufferLength

Syntax	SetBufferLength(hPosys, bufferID, length) stdcall GetBufferLength(hPosys, bufferID) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
		Type	Range	
	bufferID	unsigned 16 bit	0 to 31	
	length	unsigned 32 bit	1 to 2^{30}-1	
Definition	SetBufferLengthsetzt die Länge des Buffers im Speicherblock, identifiziert durch bufferID, in 32-Bit Elementen. Anmerkung: SetBufferLength resettet die Buffer Read und Write Indexe auf 0. GetBufferLengthliefert die Länge des spezifizierten Buffers.			
Einschränkung	Falls die spezifizierte Länge den maximal adressierbaren Speicherbereich überschreitet, wird SetBufferLength nicht ausgeführt und liefert einen Host-I/O Fehlercode 7 zurück (buffer bound exceeded). Anmerkung: Wenn die Bufferlänge über das Ende des maximal zur Verfügung stehen Speicherbereiches gesetzt wird, kann dies zu einem unerwarteten Reset der POSYS® oder des Rechners während der Operation führen. Es liegt daher in der Verantwortur des Anwenders Speicherbereiche sauber zu definieren.			
siehe	Set/GetBufferReadIndex, Set/GetBufferStart, Set/GetBufferWriteIndex			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetBufferLength(hPosys#, bufferID, length) GetBufferLength(hPosys#, bufferID)			

¤

Set/GetBufferReadIndex

SetBufferReadIndex
GetBufferReadIndex

Syntax	SetBufferReadIndex(hPosys, bufferID, index) stdcall GetBufferReadIndex(hPosys, bufferID) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
		Type		
	bufferID	unsigned 16 bit	Range	0 to 31
			Scaling	unity
			Units	-
	index	unsigned 32 bit	Range	0 to buffer length - 1
			Scaling	unity
			Units	double words
Definition	SetBufferReadIndexsetzt die Adresse des ReadIndex für den spezifizierten Buffer. Falls der ReadIndex auf eine Adresse ausserhalb des verfügbaren Speicherbereiches gesetzt wird, wird d Kommando nicht ausgeführt und einen Fehlercode 7 (buffer bound exceeded) zurückliefern. GetBufferReadIndexliefert den aktuellen ReadIndex für den spezifizierten Buffer.			
Einschränkung	Falls der ReadIndex ausserhalb des zulässigen Bereiches liegt, wird das Kommando nicht ausgeführt und ein Fehlercode 7 (buffer bound exceeded) zurückmmelden.			
siehe	Set/GetBufferLength, Set/GetBufferStart, Set/GetBufferWriteIndex			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetBufferReadIndex(hPosys#, bufferID, index) GetBufferReadIndex(hPosys#, bufferID)			

¤

Set/GetBufferStart

SetBufferStart
GetBufferStart

Syntax	SetBufferStart(hPosys, bufferID, address) stdcall GetBufferStart(hPosys, bufferID) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
		Type	Range	
	bufferID	unsigned 16 bit	0 to 31	
	address	unsigned 32 bit	200h to 2^{31}-1 für POSYS® 700/800/800-B/900 0h to 2^{31}-1 für POSYS® 1800/1900	
Definition	SetBufferStartsetzt die Startadresse für den spezifizierten Buffer. Die BufferStartAdresse für POSYS® 700/800/800-B/900 muss 200h oder höher sein ansonsten kann es 0h sein Anmerkung: SetBufferStart setzt die Read und Write Indexe auf 0. GetBufferStartliefert die Startadresse für den spezifizierten Buffer.			
Einschränkung	Falls die spezifizierte Länge den maximal adressierbaren Speicherbereich überschreitet, wird SetBufferStart nicht ausgeführt und liefert einen Host-I/O Fehlercode 7 zurück (buffer bound exceeded). Anmerkung: Wenn die Bufferlänge über das Ende des maximal zur Verfügung stehen Speicherbereiches gesetzt wird, kann dies zu einem unerwarteten Reset der POSYS® oder des Rechners während der Operation führen. Es liegt daher in der Verantwortur des Anwenders Speicherbereiche sauber zu definieren.			
siehe	Set/GetBufferLength, Set/GetBufferReadIndex, Set/GetBufferWriteIndex			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetBufferStart(hPosys#, bufferID, address) GetBufferStart(hPosys#, bufferID)			

¤

Set/GetBufferWriteIndex

SetBufferWriteIndex
GetBufferWriteIndex

Syntax	SetBufferWriteIndex(hPosys, bufferID, index) stdcall GetBufferWriteIndex(hPosys, bufferID) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
		Type		
	bufferID	unsigned 16 bit	Range	0 to 32
			Scaling	unity
			Units	-
	index	unsigned 32 bit	Range	0 to buffer length - 1
			Scaling	unity
			Units	double words
Definition	SetBufferWriteIndex setzt die Adresse des WriteIndex für den spezifizierten Buffer. Falls der WriteIndex auf eine Adresse außerhalb der Länge des Buffers gesetzt wird, wird das Kommando nicht ausgeführt und ein Fehler zurückgeliefert. GetBufferWriteIndex liefert den aktuellen WriteIndex für den spezifizierten Buffer.			
Einschränkung	Falls der Read Index auf eine Adresse ausserhalb der spezifizierten Bufferlänge liegt, wird das Kommando nicht ausgeführt und ein HostIOError 7, buffer bound exceeded, zurückgemeldet.			
siehe	Set/GetBufferLength, Set/GetBufferReadIndex, Set/GetBufferStart			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetBufferWriteIndex(hPosys#, bufferID, index) GetBufferWriteIndex(hPosys#, bufferID)			

¤

WriteBuffer

Syntax	WriteBuffer(hPosys, bufferID, value) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	POSYS® 18xx, 19xx	Type	Range	
	bufferID	unsigned 16 bits	0 to 31	
	value	signed 32 bits	-2^{31} to $2^{31}-1$	
	POSYS® 7xx, 8xx, 8xx-B, 9xx	Type	Range	
	bufferID	unsigned 16 bits	0 to 15	
	value	signed 32 bits	-2^{31} to $2^{31}-1$	
Definition	WriteBufferschreibt ein 32-Bit Wert in die aktuelle Position des spezifizierten Buffers. Die aktuelle Position wird bestimmt indem die Basisadresse (SetBufferStart Kommando) dem aktuellen BufferWriteIndex (SetBufferWriteIndex Kommando) hinzu addiert wird. Nachdem der Inhalt gelesen wurde, wird der WriteIndex um "1" inkrementiert; falls das Ergebnis gleich der Bufferlänge entspricht (SetBufferLength Kommando), wird der WriteIndex = 0. Einige Operationen der POSYS® verändern automatisch den WriteIndex, wie z.B. ein Trace.			
Einschränkung				
siehe	ReadBuffer, Set/GetBufferWriteIndex			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	WriteBuffer(hPosys#, bufferID, value)			

¤

WriteDPRamLong

nur gültig für POSYS® 18xx/19xx mit Dual Port RAM

Syntax	WriteDPRamLong(hPosys,offset,value) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	offset	unsigned 32 Bit	Speicherbasisadresse + offset	
	value	signed 32 Bit	Range	-2^{31} to $2^{31}-1$
			Scaling	Unity
			Units	-
Ergebnis				
Definition	Schreibt einen 32-Bit Wert zur Speicherbasisadresse + Offset			
Einschränkung	Dieses Kommando ist nur für POSYS® Karten mit Dual Port RAM verfügbar.			
siehe	ReadDPRamLong, EnableDPRam, DisableDPRam			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	WriteDPRamLong(hPosys#,offset,value)			

¤

Motor Output

GetActiveMotorCommand

GetActiveMotorCommand

nur gültig mit POSYS® 1800/1900 falls mit Motion Control Chipsatz bestückt > V2.x

Syntax	GetActiveMotorCommand(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
Ergebnis	Name	Type		
	Motor output command	signed 16 bits	Range	-2^{15} to $2^{15}-1$
			Scaling	$100/2^{15}$
			Units	% output
Definition	GetActiveMotorCommand liefert den gegenwärtig ausgegebenen Wert für die Drehmomentsollwertvorgabe für die spezifizierte Achse. Dies ist der Eingangswert zur Kommutierung. Seine Quelle hängt sowohl vom Motortyp als auch vom Operating Mode der Ach ab. Für bürstenbehaftete und bürstenlose Servomotoren: falls Position Loop aktiviert ist, ist es die Ausgabe für den Position Loop Filter. falls der Trajectory Generator ohne Position Loop aktiv ist, ist es die Ausgabe des Trajectory Generator's. Falls beide deaktiviert sind ist es die Ausgabe des Inhaltes des Motorausgaberegisters. Für Mikroschrittmotoren: ist es die Ausgabe des Inhaltes des Motorausgaberegisters mit Berücksichtigung des Current Holding Registers. Skalierungsbeispiel: um den zurückgelieferten Wert in Prozente auszugeben, multiplizieren Sie 100 / 32768. Z.B. falls der Wert -123 ist, entspricht dies -123*(100 / 32768) oder -0,3754% de maximal möglichen Ausgabe.			
Einschränkung				
siehe	Set/GetMotorCommand			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	GetActiveMotorCommand(hPosys#, axis)			

¤

GetCurrentMotorCommand

GetCurrentMotorCommand

Syntax	GetCurrentMotorCommand(hPosys, axis) stdcall		
Motortyp	DC Brush	DC Brushless	Microstepping
Argumente	Name	Instance	Encoding
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice
	axis	Axis1 Axis2	0 1

		Axis3	2	
		Axis4	3	
Ergebnis	Name	Type		
	Motor output command	signed 16 bits	Range	-2^{15} to $2^{15}-1$
			Scaling	$100/2^{15}$
			Units	% output

Definition	GetCurrentMotorCommand liefert den gegenwärtig ausgegebenen Wert für die Drehmomentsollwertvorgabe für die spezifizierte Achse. In C losed-Loop-Mode ist dies die Ausg durch den Servofilter; in Open-Loop-Mode ist es die Ausgabe durch den Motor Output C ommar Register. Skalierungsbeispiel: um den zurückgelieferten Wert in Prozente auszugeben, multiplizieren Sie 100 / 32768. Z.B. falls der Wert -123 ist, entspricht dies -123*(100 / 32768) oder -0,3754% de maximal möglichen Ausgabe.
Einschränkung	
siehe	Set/GetMotorCommand
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)
MotionScript® Aufruf-konvention	GetCurrentMotorCommand(hPosys#, axis)

¤

© POSYS Motion Control GmbH & Co.KG, 2013

SetCurrentHolding
GetCurrentHolding

nur gültig mit POSYS® 1800/1900 falls mit Motion Control Chipsatz bestückt > V2.x

Syntax	SetCurrentHolding(hPosys, axis, parameter, value) stdcall GetCurrentHolding(hPosys, axis, parameter) stdcall			
Motortyp			Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	parameter	Holding Motor Limit Holding Delay	0 1	
	value	unsigned 16 bits	Range	0 to 2^{15-1}
			Scaling	$100/2^{15}$
			Units	% output
Definition	SetCurrentHolding konfiguriert das Haltemoment. Die beiden zu setzenden Parameter sind das *Holding Motor Limit*, der maximale Haltestrom und *Holding Delay*, die Anzahl Zyklen die gewartet werden soll nach Bewegungsende ehe in den Haltemodus geschaltet werden soll. Das Holding Motor Limit wird in % des maximalen Motorstromes angegeben. Die Skalierung ist i Werten von $100/2^{15}$ definiert und der Bereich ist von 0 bis 2^{15-1}. Es definiert den Wert zu dem der Haltestrom begrenzt wird wenn der Motor sich im Stillstand befindet. Diese Begrenzung wird als zusätzlicher Begrenzungswert zum Motor Limit hinzugefügt, dadurch wird das niedrigere der beiden Werte aktiv wenn im Haltestatus. Der Parameter *Holding Delay* ist in Einheiten von Zyklen mit Einheitenskalierung und einem Berei von 0 bis 2^{15-2}. Es definiert die Wartezeit ab Beendigung einer Bewegung bis wann in den *Holding Current Limit* gewechselt wird. Das bedeutet es gibt eine Verzögerung von n-Zyklen nach *Motion Complete* bis das *In Holding Bit* im Drive Status Register gesetzt und die Motorsollwertausgabe vom *Holding Motor Limit* begrenzt wird. Falls der Holding Delay auf 2^{15-1} gesetzt (Standard) ist, wird die Achse nie in den Haltemoment wechseln. GetCurrentHolding liefert die gegenwärtig programmierten Werte.				
Einschränkung	Für Schrittmotoren mit Takt & Rchtungssignalausgabe funktioniert nur der Parameter *Holding Delay* . Es definiert die Anzahl Zyklen ehe das AtRest-Signal aktiviert wird nach *Motion Complete*.			
siehe	GetDriveStatus, Set/GetSampleTime			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetCurrentHolding(hPosys#, axis, parameter, value) GetCurrentHolding(hPosys#, axis, parameter)			

¤

Set/GetMotorBias

SetMotorBias
GetMotorBias

Syntax	SetMotorBias(hPosys, axis, bias) stdcall GetMotorBias(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless		
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	bias	signed 16 bits	Range	-2^{15} to $2^{15}-1$
			Scaling	$100/2^{15}$
			Units	% output
Definition	SetMotorBiassetzt die Offsetspannung des digitalen Servofilters für die spezifizierte Achse. GetMotorBiasliest den programmierten Wert des digitalen Servoverstärkers. Skalierungsbeispiel: Falls es gewünscht wird, dass ein Spannungsoffsetwert von -2,5% der vollen Skala auf das Ausgangssignal des Servoreglers ausgegeben wird, dann sollte dieser Register mit einem Wert -2,5*32768/100 = -819 (dezimal) geladen werden. Dies entspricht einem Hexadezimalwert von 0FCCDh.			
Einschränkung				
siehe	Set/GetMotorCommand, Set/GetMotorLimit			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetMotorBias(hPosys#, axis, bias) GetMotorBias(hPosys#, axis)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

SetMotorCommand
GetMotorCommand

Syntax	SetMotorCommand(hPosys, axis, value) stdcall GetMotorCommand(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	value	signed 16 bits	Range	-2^{15} to $2^{15}-1$
			Scaling	$100/2^{15}$
			Units	% output
Definition	SetMotorCommand lädt den Register des Motor-Command Buffers der spezifizierten Achse. Für die Mikroschrittvarianten wird dieses Kommando verwendet die Magnitude des Ausgangssignals zu kontrollieren. GetMotorCommand liest den Inhalt des Registers des Motor-Command Buffers. Skalierungsbeispiel: Falls ein Ausgabewert von 13,7% der vollen Skala als Ausgangssignal gewünscht wird, dann muß der folgende Wert in das Register geladen werden: 13,7*32768/100 = 4489 (dezimal). Dies entspricht einem Hexadezimalwert von 1189h.			
Einschränkung	SetMotorCommand ist nur gültig wenn der Motor ausgeschaltet ist und ist nur für die Servoversionen und Mikroschrittversionen anwendbar. SetMotorCommand ist ein gebuffertes Kommando. Es wird erst aktiviert nach einem Update oder MultiUpdate Kommando. Dieses Kommando ist für die Schrittmotorversionen nicht verfügbar.			
siehe	Set/GetMotorBias, Set/GetMotorLimit, Set/GetMotorMode, MultiUpdate, Update			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetMotorCommand(hPosys#, axis, value) GetMotorCommand(hPosys#, axis)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

SetMotorMode
GetMotorMode

Syntax	SetMotorMode(hPosys, axis, mode) stdcall GetMotorMode(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	mode	off on	0 1	

Definition

SetMotorModebestimmt für die spezifizierte Achse ob der Motor im C losed-Loop-Mode oder in Open-Loop-Mode betrieben wird. Wenn ON, dann ist die Achse im C losed-Loop-Mode und wird digitalen Servofilter geregelt. Bei den Schrittmotorversionen kontrolliert der Bahngenerator das Ausgangssignal. Für alle Motortypen gilt, wenn EncoderSource auf inkrementell oder parallel gesetzt ist, wird der Schleppabstandsfehler gelöscht, wie bei C learPositionError.

Wenn OFF, dann ist der Motor im Open-Loop-Mode, und wird durch Kommandos, die direkt in da Motorsignalausgangsregister, durch den Host geschrieben werden, betrieben. Das Setzen des MotorModes auf OFF resettet auch den Bahngenerator und jede aktive Bewegung wird zu einem abrupten Stopp gebracht. Zusätzlich wird die maximale Geschwindigkeit auf Null gestellt. Die Taktsignale für Achsen im Schrittmotor- bzw. Mikroschrittmotormodus sind ausgeschaltet.

Motor Type	Motor Mode	Motor Output Source
DC brush; brushless DC	off	Motor command register
	on	servo filter
Pulse & direction; microstepping	off	N/A
	on	Trajectory generator

GetMotorMode liest den programmierten Modus.

Einschränkung	
siehe	GetActivityStatus, Set/GetMotorCommand
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)
MotionScript® Aufruf-konvention	SetMotorMode(hPosys#, axis, mode) GetMotorMode(hPosys#, axis)

¤

Set/GetMotorType

SetMotorType
GetMotorType

nur gültig für POSYS® 18xx/19xx

Syntax	SetMotorType(hPosys, axis, type) stdcall GetMotorType(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4		0 1 2 3
	type	Brushless DC (3 phase) Brushless DC (2 phase) Microstepping (3 phase) Microstepping (2 phase) Pulse & Direction DC Brush		0 1 2 3 4 7

Definition

SetMotorType bestimmt den Typ des Motors, der an der spezifizierten Achse angeschlossen ist. Dieses Kommando bestimmt die Anzahl Phasen für die Kommutierung an dieser Achse, als auch die Konfiguration für den verwendeten Motortyp. Für weitere Informationen verweisen wir hier auf das Benutzerhandbuch für die POSYS®. Die Tabelle unten listet für jeden Motortyp die Anzahl verwendeter Phasen die kommutiert werden.

GetMotorType liefert den programmierten Motortyp zurück.

Motor Type	Kommutierung
DC Brushless (3 phase)	3 phase
DC Brushless (2 phase)	2 phase
Microstepping (3 phase)	3 phase
Microstepping (2 phase)	2 phase
Pulse & Direction	None
DC Brushed	None

Einschränkung

nur gültig für POSYS® 18xx/19xx.

Die Art des verwendeten Motors sollte nur einmal gesetzt werden; entweder während des Einschaltens durch das Konfigurationswort (EPROM) oder sofort nach einem Reset mit dem Kommando SetMotorType. Ist es einmal gesetzt, sollte es nicht verändert werden.

siehe

Set/GetOutputMode

DLL: Delphi, C++, VB

wir verweisen auf die verschiedenen Importdateien (imp*.*)

MotionScript® Aufruf-konvention

SetMotorType(hPosys#, axis, type)
GetMotorType(hPosys#, axis)

¤

Set/GetOutputMode

SetOutputMode
GetOutputMode

Syntax	SetOutputMode(hPosys, axis, mode) stdcall GetOutputMode(hPosys, axis) stdcall		
Motortyp	DC Brush	DC Brushless	Microstepping
Argumente	Name	Instance	Encoding
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3
	mode	Bipolar DAC PWMSignMagnitude PWM5050Magnitude SPIDAC Unipolar DAC	0 1 2 3 (nur POSYS® 18xx/19xx) 4 (nur POSYS® 18xx/19xx)
Definition	SetOutputModebestimmt die Art des Ausgangssignals für die spezifizierte Achse. GetOutputModeliest den Code für die aktuelle Einstellung.		
Einschränkung			
siehe	Set/GetMotorType		
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)		
MotionScript® Aufruf-konvention	SetOutputMode(hPosys#, axis, mode) GetOutputMode(hPosys#, axis)		

¤

Set/GetPWMFrequency

SetPWMFrequency
GetPWMFrequency

nur gültig für POSYS® 18xx/19xx

Syntax	SetPWMFrequency(hPosys, axis, frequency) stdcall GetPWMFrequency(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	frequency	unsigned 16 bits	Range	0 to 2^{16}-1
			Scaling	$1/2^8$
			Units	kHz
Definition	SetPWMFrequencysetzt die Frequenz für das impuls-breiten-modulierte Signal für die spezifizierte Achse. Zwei Frequenzen können eingestellt werden und werden in der Tabelle unte beschrieben. Um eine der beiden Frequenzen auszuwählen, muss das entsprechende Argument der Tabelle ausgewählt und an die POSYS® gesendet werden. GetPWMFrequency liefert den programmierten Wert zurück.			
	Frequenz	PWM Bit Auflösung	Aktuelle Frequenz	SetPWM-Frequency Wert
	20 kHz	10 bit	19.531 kHz	5
	80 kHz	8 bit	78.124 kHz	20
Einschränkung	Nur 20 kHz und 80 kHz werden derzeit von der POSYS® unterstützt. Die PWM Frequenz kann nu während der Motor still steht verändert werden. Dieses Kommando ist nicht verfügbar in den POSYS® 1800/1900 Versionen mit Piezo-Funktionalität. Die PWM Frequenz ist fix auf 20 kHz/10-bit Auflösung eingestellt.			
siehe	Set/GetOutputMode			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetPWMFrequency(hPosys#, axis, frequency) GetPWMFrequency(hPosys#, axis)			

¤

Set/GetStepRange

SetStepRange
GetStepRange

Syntax	SetStepRange(hPosys, axis, frequency) stdcall GetStepRange(hPosys, axis) stdcall		
Motortyp			Pulse & Direction
Argumente	Name	Instance	Encoding
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3
	frequency	5 MHz 625 KHz 156.25 KHz 39.062 KHz	1 4 6 8
Definition	SetStepRangesetzt die maximale Taktfrequenz für die spezifizierte Achse. z.B. falls der gewünschte maximale Takt = 200000 Takte/Sek. ist, dann muss dem Kommando SetStepRange der Wert 4 übergeben werden (siehe obige Listung der Werte). GetStepRangeliest den programmierten Wert zurück.		
Einschränkung	Dieses Kommando ist nur für Achsen im Schrittmotormodus verfügbar.		
siehe			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)		
MotionScript® Aufruf-konvention	SetStepRange(hPosys#, axis, frequency) GetStepRange(hPosys#, axis)		

¤

Operaing Mode, Event und Update Control

GetActiveOperatingMode

nur gültig mit POSYS® 1800/1900 falls mit Motion Control Chipsatz bestückt > V2.x

Syntax	GetActiveOperatingMode(hPosys, axis) stdcall		
Motortyp	DC Brush	DC Brushless	
Argumente	Name	Instance	Encoding
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3
Ergebnis	Name	Type	
	mode	unsigned 16 bits	bit field
Definition	GetActiveOperatingMode liefert den aktuellen OperatingMode der spezifizierten Achse. Dies ka muss aber nicht der gleiche Mode sein wie der Static Operating Mode, da sicherheitsabhängige oder programmierbare Bedingungen den Active Operating Mode verändern können. Falls unterschiedlich, kann der Static Operating Mode wiederhergestellt werden durch Aufruf des Kommandos RestoreOperatingMode. Im Folgenden sind die Bit-Definitionen gelistet.		
	Name	Bit	Beschreibung
	Axis Enabled	0	0: keine Achse aktiv, Achsenausgaben im Resetstatus 1: Achse aktiv
	Motor Output Enabled	1	0: Achsenmotorausgabe deaktiviert 1: Achsenmotorausgabe aktiviert
	Reserved	2-3	
	Position Loop Enabled	4	0: Achsen Position Loop deaktiviert 1: Achsen Position Loop aktiv
	Trajectory Enabled	5	0: Trajectory deaktiviert 1: Trajectory aktiviert
	Reserved	6-15	
	Falls die Achse deaktiviert ist, erfolgt keine Ausgabe und auch sonst wird keine Aktivität erfolge (Enkoderzählung, AxisIn, AxisOut). Falls nur die Motorausgabe deaktiviert ist, sind alle anderen Funktionen aktiv. Falls ein Loop deaktivert ist, werden die entsprechenden Loops umgangen und alle statischen Variablen werden gelöscht. Falls der Trajectory deaktiviert ist, funktioniert die Ac mit Geschwindigkeit = 0.		
Einschränkung			
siehe	Set/GetOperatingMode, RestoreOperatingMode, Set/GetEventAction, Set/GetBreakpoint		
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)		
MotionScript® Aufruf-konvention	GetActiveOperatingMode(hPosys#, axis)		

¤

MultiUpdate

Syntax	MultiUpdate(hPosys, mask) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	mask	None Axis1mask Axis2mask Axis3mask Axis4mask	0 1 2 4 8	
Definition	MultiUpdateführt ein Update auf allen Achsen aus, deren entsprechender Bit im Maskparamete auf 1 gesetzt ist. Nachdem dieses Kommando für jene im Maskparameter ausgewählten Achsen ausgeführt wurde, werden alle gebufferten Datenparameter in die entsprechenden Run-Time Register kopiert. Die folgenden Bahnparameterkommandos sind gebuffert: Acceleration, Deceleration, GearRatio, Jerk, Position, ProfileMode, StopMode, und Velocity. Die folgenden PID Filterparameterkommandos sind gebuffert: DerivativeTime, IntegrationLimit, Kaff, Kd, Ki, Kp, und Kvff. Folgendes Kommando für die Motorausgabe ist gebuffert: MotorCommand			
Einschränkung				
siehe	Update			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	MultiUpdate(hPosys#, mask)			

¤

RestoreOperatingMode

nur gültig mit POSYS® 1800/1900 falls mit Motion Control Chipsatz bestückt > V2.x

Syntax	RestoreOperatingMode(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
Definition	RestoreOperatingMode wird verwendet um die spezifierte Achse wieder in den static Operating Mode zurück zu versetzen. Es sollte verwendet falls der active Operating Mode aufgrund von Ereignissen (*programmierte* oder *safety*) verändert wurde. Der Aufruf von RestoreOperatingMode aktivert alle Loops die vorher deaktiviert wurden aufgrund von Ereignissen.			
Einschränkung	Im Event Status Register sollten alle Bits gelöscht werden ehe mit RestoreOperatingMode wieder in den static Operating Mode zurückgeschaltet wird. Falls ein Bit im Event Status Register welches eine Änderung im Operating Mode verursacht hat, nicht gelöscht wurde ehe das Kommando ausgeführt wurde, wird ein Fehler zurückgemeldet. Eine Ausnahme bildet das Argument *Motion Error*, welches nicht gelöscht werden muss vor Ausführung des Kommandos.			
siehe	GetActiveOperatingMode, Set/GetOperatingMode, Set/GetEventAction, Set/GetBreakpoint			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	RestoreOperatingMode(hPosys#, axis)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

SetEventAction
GetEventAction

nur gültig mit POSYS® 1800/1900 falls mit Motion Control Chipsatz bestückt > V2.x

Syntax	SetEventAction(hPosys, axis, event, action) stdcall GetEventAction(hPosys, axis, event) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	event	Immediate Positive Limit Negative Limit Motion Error	0 1 2 3	
	action	none Reserved Abrupt Stop Smooth Stop Reserved Disable Position Loop & Higher Modules n/a Disable Motor Output & Higher Modules Abrupt Stop with Position Error Clear	0 1 2 3 4 5 6 7 8	
Definition	SetEventAction bestimmt welche Aktionen eine Achse ausführt aufgrund eines *events*. action kann sein, dass der Operating Mode modifiziert wird indem einige oder alle Loops deaktivirt werden, oder im Falle dass alle Loops eingeschaltet bleiben einen Abrupt oder Smooth Stop ausführen. Der Abrupt Stop kann mit oder ohne Löschung des Schleppabstandsfehlers geschehen. Falls eines der *events* des Kommandos SetEventAction ein *action* auslöst, musst das event Bit im Event Status Register gelöscht werden, ehe wieder zum normalen Betrieb zurückgekehrt werden kann. Im Falle von Trajectory Stops bedeutet dies dass zuerst das Bit gelöscht werden muss, eine neue Bewegung ausgeführt werden kann. Für Änderungen im Operating Mode bedeutet es dass das Bit gelöscht werden muss ehe Operating Mode durch die beiden Kommandos RestoreOperatingMode oder SetOperatingMode wieder aktiviert werden kann. Eine Ausnahme bildet das Motion Error *event*, welches nur im Event Status gelöscht werden muss falls *action* Abrupt Stop oder Smooth Stop war. Falls es aber Änderungen im Operating Mode herbeiführt, kann der Operating Mode wiederhergestellt werden ohne Event Status zuerst das Bit zu löschen. GetEventAction liest den programmierten Status zurück. Nur im Falle von *event* = Immediate funktioniert das Zurücklesen nicht.			
Einschränkung	Falls ein Smooth Stop *action* ausgelöst wird während eine S-Kurve abgefahren wird, kann der Trajectory nicht wieder gestartet werden ehe der Smooth Stop vollendet wurde. Falls ein Smooth Stop *action* ausgelöst wird während eine im Electronic Gearing ausgeführten Bewegungsprofiles, wird ein Abrupt Stop ausgeführt.			
siehe	GetActiveOperatingMode, RestoreOperatingMode, Set/GetOperatingMode			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetEventAction(hPosys#, axis, event, action) GetEventAction(hPosys#, axis, event)			

¤

Set/GetOperatingMode

SetOperatingMode
GetOperatingMode

nur gültig mit POSYS® 1800/1900 falls mit Motion Control Chipsatz bestückt > V2.x

Syntax	SetOperatingMode(hPosys, axis, mode) stdcall GetOperatingMode(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	mode	unsigned 16 bit	see below	

Definition

SetOperatingMode bestimmt wie die Achse betrieben wird. Jedes Bit des Parmeters *mode* konfiguriert je nachdem ob eine Funktion oder Loop aktiviert oder deaktiviert wird wie unten beschrieben.

Name	Bit	Beschreibung
Axis enabled	0	0: Achse nicht aktiv, Achsenwerte auf Default eingestellt 1: Achse aktiv
Motor Output Enabled	1	0: Motorausgabe deaktiviert 1: Motorausgabe aktiviert
n/a	2	
Reserved	3	Reserved
Position Loop Enabled	4	0: Positon Loop ignoriert 1: Position Loop aktiv
Trajectory Enabled	5	0: Trajectory Generator deaktiviert 1: Trajectory Generator aktiviert
Reserved	6-15	Reserved

Falls die Achse deaktiviert ist, erfolgt keine Reaktion. Die Achsenausgabe ist auf seinen Einschaltzustand zurückgesetzt. Falls die Motorausgabe deaktiviert ist, funktioniert die Achse normal mit der Ausnahme, dass keine Motorausgabe erfolgt. Falls ein Loop deaktivert ist, funktioniert die Achse normal, nur dass die Eingänge für den deaktivierten Loop übergangen werden und die Ausgabe direkt erfolgt, während alle internen Statusvariablen gelöscht werden (z.B. Integrator Sum etc.). Falls der Trajectory Generator deaktiviert ist, funktioniert die Achse aber die Geschwindigkeit ist = 0.

Falls die Achse im Drehmomentmodus arbeiten soll (Trajectory und Position Loop deaktiviert), wird der Operating Mode mit dem Wert 0007h programmiert.

Dieses Kommando sollte verwendet werden den statischen Operating Mode einer Achse zu konfigurieren. Die aktuelle Konfiguration kann agefragt werden mit dem Kommando GetActiveOperatingMode. GetOperatingMode liefert immer den mit SetOperatingMode programmierten Wert zurück. Die Ausführung von SetOperatingMode konfiguriert beides, den aktiven OperatingMode und den statischen Operating Mode, mit den gewüschten Werten.

GetOperatingMode liest den programmierten Modus von *axis*.

Einschränkung	
siehe	GetActiveOperatingMode, RestoreOperatingMode
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)
MotionScript® Aufruf-	SetOperatingMode(hPosys#, axis, mode) GetOperatingMode(hPosys#, axis)

konvention

¤

© POSYS Motion Control GmbH & Co.KG, 2013

SetUpdateMask
GetUpdateMask

nur gültig mit POSYS® 1800/1900 falls mit Motion Control Chipsatz bestückt > V2.x

Syntax	SetUpdateMask(hPosys, axis, mask) stdcall GetUpdateMask(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type	Scaling	
	mask	unsigned 16 bits	bitmask	
Definition	SetUpdateMaskbestimmt welche Loops upgedated werden aufgrund einer Updateaktion. Falls ein bitmask für ein bestimmter Loop im *mask* gesetzt wurde, werden die Parameter für diesen Loop mit den gebufferten Werten im Register ersetzt, sobald ein Update oder MultiUpdate Kommando erfolgt.			
	Name	Bit(s)	Beschreibung	
	Trajectory	0	auf 1 setzen um die Trajectory mit gebufferten Parametern upzudaten	
	Position Loop	1	auf 1 setzen um Position Loop mit gebufferten Parametern upzudaten	
	Reserved	2-15		
	GetUpdateMaskliefert die programmierte Maske zurück.			
Einschränkung				
siehe	Set/GetBreakpointUpdateMask, Update, MultiUpdate			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetUpdateMask(hPosys#, axis, mask) GetUpdateMask(hPosys#, axis)			

¤

Update

Syntax	Update(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
Definition	Update führt ein Update auf allen Achsen aus, deren entsprechender Bit im Maskparameter auf gesetzt ist. Nachdem dieses Kommando für jene im Maskparameter ausgewählten Achsen ausgeführt wurde, werden alle gebufferten Datenparameter in die entsprechenden Run-Time Register kopiert. Die folgenden Bahnparameterkommandos sind gebuffert: Acceleration, Deceleration, GearRatio, Jerk, Position, ProfileMode, StopMode, und Velocity. Die folgenden PID Filterparameterkommandos sind gebuffert: DerivativeTime, IntegrationLimit, Kaff, Kd, Ki, Kp, und Kvff. Folgendes Kommando für die Motorausgabe ist gebuffert: MotorCommand			
Einschränkung				
siehe	MultiUpdate			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	Update(hPosys#, axis)			

¤

Profilerstellung

GetCommandedAcceleration

GetCommandedAcceleration

Syntax	GetCommandedAcceleration(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
Ergebnis	Name	Type		
	acceleration	signed 32 bits	Range	-2^{31} to $2^{31}-1$
			Scaling	$1/2^{16}$
			Units	counts/cycle² steps/cycle²
Definition	GetCommandedAcceleration liefert für die spezifizierte Achse den aktuellen Wert für die Sollbeschleunigung zurück. Die Sollbeschleunigung ist die Beschleunigung die vom Regler ausgegeben wird. Skalierungsbeipsiel: Wenn ein Wert von 114688 zurückgeliefert wird, dann entspricht dies einer Beschleunigung von 114688 / 65536 = 1750 counts/cycle²			
Einschränkung	Dieses Kommando funktioniert in den Profilmodi Trapez, S-Kurve oder Velocity Contouring. Es funktioniert nicht im Electronic Gearing Mode.			
siehe	GetCommandedPosition, GetCommandedVelocity			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	GetCommandedAcceleration(hPosys#, axis)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

GetCommandedPosition

Syntax	GetCommandedPosition(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
Ergebnis	Name	Type		
	position	signed 32 bits	Range	-2^{31} to $2^{31}-1$
			Scaling	unity
			Units	counts steps
Definition	GetCommandedPosition liefert für die spezifizierte Achse die momentane Sollposition zurück. I Sollposition ist der momentane tatsächliche Wert der ausgegeben wird vom Regler. Diese Funktion ist in allen Profilmodi verwendbar.			
Einschränkung				
siehe	GetCommandedAcceleration, GetCommandedVelocity			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	GetCommandedPosition(hPosys#, axis)			

¤

GetCommandedVelocity

Syntax	GetCommandedVelocity(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
Ergebnis	Name	Type		
	velocity	signed 32 bits	Range	-2^{31} to $2^{31}-1$
			Scaling	$1/2^{16}$
			Units	counts/cycle Steps/cycle
Definition	GetCommandedVelocity liefert für die spezifizierte Achse die momentane Sollgeschwindigkeit zurück. Die Sollgeschwindigkeit ist der momentane tatsächliche Wert der ausgegeben wird vom Regler. Skalierungsbeispiel: falls der Wert 1234567 empfangen wird (FFEDh in high-Word, 2979H in low-Word) dann entspricht dies einer Geschwindigkeit von 1234567/ 65536 = 18,838 counts/cy Diese Funktion ist in allen Profilmodi verwendbar.			
Einschränkung				
siehe	GetCommandedAcceleration, GetCommandedPosition			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	GetCommandedVelocity(hPosys#, axis)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

SetAcceleration
GetAcceleration

Syntax	SetAcceleration(hPosys, axis, acceleration) stdcall GetAcceleration(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	Name	Type		
	acceleration	unsigned 32 bit	Range	0 to 2^{31}-1
			Scaling	$1/2^{16}$
			Units	counts/cycle² steps/cycle²
Definition	SetAcceleration lädt für die spezifizierte Achse die maximale positive Beschleunigung in das Register. Dieses Kommando ist verwendbar in den Profilmodi: trapezförmig, S-Kurve und Geschwindigkeitsmodus. GetAcceleration liest für die spezifizierte Achse die programmierte Beschleunigung, die mit der Kommando SetAcceleration gesetzt wurde. Skalierungsbeispiel: Um einen Wert von 1,750 counts/cycle² zu laden, multiplizieren Sie mit 65 (Ergebnis 114688) und laden Sie diesen Wert als 32-Bit Integer. Übergeben Sie an das höherwertige Wort 0001h und das niederwertige Wort C 000h. Zurückgelieferte Werte mit GetAcceleration müssen entsprechend mit 65536 dividiert werden um sie in Einheiten von counts/cycle² zu konvertieren.			
Einschränkung	Eine Achse die im S-Kurven Modus und in Bewegung ist kann keine neuen Werte mit SetAcceleration übernehmen. SetAcceleration ist nicht anwendbar im Electronic Gearing Profil Modus. SetAcceleration ist ein gebuffertes Kommando. Ein Update bzw. MultiUpdate Kommando ist notwendig um die neuen Werte zu aktivieren.			
siehe	Set/GetDeceleration, Set/GetJerk, Set/GetPosition, Set/GetVelocity, MultiUpdate, Update			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetAcceleration(hPosys#, axis, acceleration) GetAcceleration(hPosys#, axis)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

SetDeceleration
GetDeceleration

Syntax	SetDeceleration(hPosys, axis, deceleration) stdcall GetDeceleration(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	deceleration	unsigned 32 bits	Range	0 to 2^{31}-1
			Scaling	$1/2^{16}$
			Units	counts/cycle² steps/cycle²
Definition	SetDeceleration lädt für die spezifizierte Achse die maximale negative Beschleunigung in das Register. GetDeceleration liest für die spezifizierte Achse die programmierte negative Beschleunigung, d mit dem Kommando SetDeceleration gesetzt wurde. Skalierungsbeispiel: Um einen Wert von 1,750 counts/cycle² zu laden multiplizieren Sie mit 655 (Ergebnis 114688) und laden Sie diesen Wert als 32-Bit Integer. Übergeben Sie an das höherwertige Wort 0001h und das niederwertige Wort C 000h. Zurückgelieferte Werte mit GetDeceleration müssen entsprechend mit 65536 dividiert werden um sie in Einheiten von counts/cycle² zu konvertieren.			
Einschränkung	SetDeceleration ist ein gebuffertes Kommando. Der neue Wert wird nicht aktualisiert ehe ein Update oder MultiUpdate Kommando ausgeführt wurde. Eine Achse die im S-Kurven Modus und in Bewegung ist, kann keine neuen Werte mit SetDeceleration übernehmen. SetDeceleration ist nicht anwendbar im Electronic Gearing Profil Modus.			
siehe	Set/GetAcceleration, Set/GetPosition, Set/GetVelocity, MultiUpdate, Update			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetDeceleration(hPosys#, axis, deceleration) GetDeceleration(hPosys#, axis)			

¤

Set/GetGearMaster

SetGearMaster
GetGearMaster

Syntax	SetGearMaster(hPosys, axis, masterAxis, source) stdcall GetGearMaster(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	masterAxis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	source	Actual Commanded	0 1	
Definition	SetGearMasterdefiniert die Slaveachse(n) und die Masterachse für das Electronic Gearing Prof und definiert die Quelle, Ist- oder Sollposition der Masterachse, die als Vorgabe für die Slaveacl verwendet werden soll. "masterAxis" bestimmt welche Achse den Slaveachsen als Quellinformation dient. Mindestens 2 Achsen müssen aktiviert sein (SetAxisMode Kommando). "source" bestimmt ob die Sollposition Masterachse, vorgegeben durch den Servoregler, als Referenz verwendet wird um die Slaveach anzutreiben, oder ob die Istposition (Enkoderposition) der Masterachse als Referenz für den Antrieb der Slaveachse dienen soll. GetGearMasterliest den Code der eingestellten Achsen und die Positionsquelle.			
Einschränkung	Um den Electronic Gear Mode verwenden zu können, muss die MasterAxis aktiviert sein. Der ProfileMode muss entsprechend auf "Electronic Gearing" eingestellt sein.			
siehe	Set/GetGearRatio, Set/GetProfileMode, MultiUpdate, Update			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetGearMaster(hPosys#, axis, masterAxis, source) GetGearMaster(hPosys#, axis)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

SetGearRatio
GetGearRatio

Syntax	SetGearRatio(hPosys, slaveAxis, ratio) stdcall GetGearRatio(hPosys, slaveAxis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	slaveAxis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	ratio	signed 32 bits	Range	-2^{31} to $2^{31}-1$
			Scaling	$1/2^{16}$
			Units	Slavecounts/ Mastercounts

Definition

SetGearRatiodefiniert das Verhältnis der Folgegeschwindigkeit von Masterachse und Slaveachs im Electronic Gearing Profile Mode. Positive Verhältnisse veranlassen die Slaveachse in die gleic Richtung zu laufen wie die Masterachse, negative Verhältnisse in die entgegengesetzte Richtung Das Verhältnis ist definierbar auf 1/65536 (gleichbedeutend mit 65536 Inkremente für die Masterachse = 1 Inkrement für die Slaveachse).

GetGearRatioliest das programmierte Folgeverhältnis für die spezifizierte Achse. Skalierungsbeispiel:

Verhältniswert	resultierendes Verhältnis
-32,768	.5 negative Slavecounts für jeden positiven Mastercount
1,000,000	15.259 positive Slavecounts für jeden positiven Mastercount
123	.0018 positive Slavecounts für jeden positiven Mastercount

Einschränkung	Dieses Kommando ist gebuffert und wird erst effektiv nach einem Update bzw. MultiUpdate Kommando.
siehe	Set/GetGearMaster, Set/GetProfileMode, MultiUpdate, Update
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)
MotionScript® Aufruf-konvention	SetGearRatio(hPosys#, slaveAxis, ratio) GetGearRatio(hPosys#, slaveAxis)

¤

Set/GetJerk

SetJerk
GetJerk

Syntax	SetJerk(hPosys, axis, jerk) stdcall GetJerk(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	jerk	unsigned 32 bits	Range	0 to 2^{31}-1
			Scaling	$1/2^{32}$
			Units	counts/cycle³ steps/cycle³
Definition	SetJerk lädt den Jerkregister in den Parameterbuffer der spezifizierten Achse. GetJerk liest den Inhalt des Jerkregisters. Skalierungsbeispiel: Um einen Jerkwert von 0,012345 counts/cycle³ zu laden (zeitliche Frequen der Änderung des Beschleunigungswertes), multiplizieren Sie mit 232 oder 4294967296. In dies Beispiel ergibt dies einen zu ladenden Wert von 53021371 (dezimal), welches dem höherwertige Wort von 0329h und dem niederwertigen Wort von 0ABBh entspricht, wenn hexadezimal übergeben.			
Einschränkung	SetJerk ist ein gebuffertes Kommando und wird erst aktiv nach einem Update oder MultiUpdate Kommando. Dieses Kommando wird nur in Verbindung mit dem S-Kurven Modus verwendet. In allen andere Modi kann es nicht verwendet werden.			
siehe	Set/GetAcceleration, Set/GetDeceleration, Set/GetPosition, Set/GetVelocity, MultiUpdate, Update			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetJerk(hPosys#, axis, jerk) GetJerk(hPosys#, axis)			

¤

Set/GetPosition

SetPosition
GetPosition

Syntax	SetPosition(hPosys, axis, position) stdcall GetPosition(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	position	signed 32 bits	Range	-2^{31} to $2^{31}-1$
			Scaling	unity
			Units	counts steps
Definition	SetPositionbestimmt die Zielposition für die spezifizierte Achse. Es wird im Trapezförmigen un S-Kurven Modus verwendet. GetPositionliest die programmierten Werte aus dem gebufferten Register.			
Einschränkung	SetPosition ist ein gebuffertes Kommando. Es wird erst aktiviert nach einem Update oder MultiUpdate Kommando.			
siehe	Set/GetAcceleration, Set/GetDeceleration, Set/GetJerk, Set/GetVelocity, GetPositionError, Set/GetPositionErrorLimit, MultiUpdate, Update			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetPosition(hPosys#, axis, position) GetPosition(hPosys#, axis)			

¤

Set/GetProfileMode

SetProfileMode
GetProfileMode

Syntax	SetProfileMode(hPosys, axis, profile) stdcall GetProfileMode(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	profile	Trapezoidal Velocity S-curve Electronic gear External	0 1 2 3 4	
Definition	SetProfileMode setzt die Profile für die spezifizierte Achse auf trapezförmig, S-Kurve, Velocity, Electronic Gearing oder External Profile Mode. GetProfileMode liest die programmierten Werte für die spezifizierte Achse aus dem gebufferten Profile Mode Register zurück.			
Einschränkung	SetProfileMode ist ein gebuffertes Kommando. Es wird erst aktiviert nach einem Update oder MultiUpdate Kommando.			
siehe	MultiUpdate, Update, Set/GetGearMaster, Set/GetGearRatio, Set/GetBufferFunction			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetProfileMode(hPosys#, axis, profile) GetProfileMode(hPosys#, axis)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

SetStartVelocity
GetStartVelocity

Syntax	SetStartVelocity(hPosys, axis, velocity) stdcall GetStartVelocity(hPosys, axis) stdcall			
Motortyp			Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	velocity	unsigned 32 bits	Range	0 to 2^{31}-1
			Scaling	$1/2^{16}$
			Units	counts/cycle steps/cycle
Definition	SetStartVelocity lädt für die spezifizierte Achse die Startgeschwindigkeit in den Buffer. GetStartVelocity liest den programmierten Wert zurück. Skalierungsbeispiel: Um eine Startgeschwindigkeit von 1750 Inkremente/Schritte pro Zyklus zu laden, multiplizieren den Wert mit 65536 (=114688) und laden Sie den entsprechenden Wert als 32-Bit Integer, dabe im höherwertigen Wort 0001 und im niederwertigen Wort C 000h übergebend. Zurückgelieferte Werte werden entsprechend zurückkonvertiert (durch 65536 dividiert werden) in Inkremente/Schritte pro Zyklus.			
Einschränkung	SetStartVelocity kann nur im Velocity Modus und Trapez Modus verwendet werden. SetStartVelocity ist ein gebuffertes Kommando. Ein Update bzw. MultiUpdate Kommando ist notwendig um die neuen Werte zu aktivieren.			
siehe	Set/GetAcceleration, Set/GetDeceleration, Set/GetVelocity, Set/GetPosition, MultiUpdate, Update			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetStartVelocity(hPosys#, axis, velocity) GetStartVelocity(hPosys#, axis)			

¤

Set/GetStopMode

SetStopMode
GetStopMode

Syntax	SetStopMode(hPosys, axis, mode) stdcall GetStopMode(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	mode	NoStop AbruptStop SmoothStop	0 1 2	
Definition	SetStopModestoppt die spezifiziete Achse. Verfügbare Stoppmethoden sind AbruptStop, welch die Achse ohne Rampe abbremst, SmoothStop, welche die Achse mit der programmierten Ramp und dem programmierten Profil (Trapez, S-Kurve etc.) abbremst, oder NoStop welches in der Regel dafür verwendet wird, ein vorhergehendes SetStopMode-Kommando auszuschalten. Anmerkung:Nach einem Update wird das gebufferte Stopp-Kommando (SetStopMode Kommando) zurück in einen NoStop Zustand resetten. In anderen Worten, wenn das Kommand SetStopMode von einem Update Kommando gefolgt wird und dann ein GetStopMode Kommando ausgeführt wird, wird der zurückgelieferte Wert NoStop sein. GetStopModeliest die Stoppmethode zurück die mit SetStopMode gesetzt wurde.			
Einschränkung	SmoothStop mode ist im Electronic Gearing Profile Mode nicht anwendbar. SetStopMode ist ein gebuffertes Kommando und wird erst aktiv nach einem Update oder MultiUpdate Kommando.			
siehe	MultiUpdate, Update			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetStopMode(hPosys#, axis, mode) GetStopMode(hPosys#, axis)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

SetVelocity
GetVelocity

Syntax	SetVelocity(hPosys, axis, velocity) stdcall GetVelocity(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	velocity	signed 32 bits	Range	-2^{31} to $2^{31}-1$
			Scaling	$1/2^{16}$
			Units	counts/cycle steps/cycle
Definition	SetVelocitylädt für die spezifizierte Achse die maximale Geschwindigkeit in den Register. GetVelocityliest den programmierten Wert. Skalierungsbeispiel: Um eine Geschwindigkeit von 1750 Inkremente pro Zyklus zu laden, multiplizieren sie mit 65536 (= 114688) und laden Sie das Ergebnis als 32-Bit Integer, dabei 00 im höherwertigen Wort und C 000h im niederwertigen Wort ubergebend. Zurückgelieferte Werte müssen entsprechend zurückkonvertiert werden.			
Einschränkung	SetVelocity kann nicht ausgeführt werden, wenn eine Achse im S-Kurven Modus ist und sich ge in Bewegung befindet. SetVelocity ist nicht im Electronic Gearing Mode verwendbar. Die Geschwindigkeit darf nicht kleiner 0 sein, mit Ausnahme im Geschwindigkeitsmodus. In dies Modus geben Werte mit Vorzeichen die Richtung vor. SetVelocity ist ein gebuffertes Kommando. Es wird erst aktiviert nach einem Update oder MultiUpdate Kommando.			
siehe	Set/GetAcceleration, Set/GetDeceleration, Set/GetJerk, Set/GetPosition, MultiUpdate, Update			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetVelocity(hPosys#, axis, velocity) GetVelocity(hPosys#, axis)			

¤

Servo Loop Kontrolle

GetTime

GetTime

Syntax	GetTime(hPosys) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
Ergebnis	Name	Type		
	time	unsigned 32 bits	Range	0 to 2^{32}-1
			Scaling	unity
			Units	cycles
Definition	Liefert die Anzahl Zyklen zurück seitdem die POSYS® initialisiert bzw. resettet wurde.			
Einschränkung				
siehe				
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	GetTime(hPosys#)			

¤

Set/GetAxisMode

SetAxisMode
GetAxisMode

Syntax	SetAxisMode(hPosys, axis, mode) stdcall GetAxisMode(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	mode	off on	0 1	
Definition	SetAxisModeaktiviert (ON) bzw. deaktiviert (OFF) die spezifizierte Achse. Eine deaktivierte Ach wird nicht auf Kommandos reagieren. GetAxisModeliefert den programmierten Zustand der spezifizierten Achse zurück.			
Einschränkung	Deaktivierte Achsen liefern keine Positionsinformationen zurück. Falls eine Rückmeldung gewünscht wird, obwohl weder Profilmodi noch PID-Regelung erfolgen soll, dann muss die Achse aktiviert bleiben.			
siehe				
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetAxisMode(hPosys#, axis, mode) GetAxisMode(hPosys#, axis)			

¤

Set/GetLimitSwitchMode

SetLimitSwitchMode
GetLimitSwitchMode

Syntax	SetLimitSwitchMode(hPosys, axis, mode) stdcall GetLimitSwitchMode(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	mode	off on	0 1	
Definition	SetLimitSwitchModeaktiviert (On) oder deaktiviert (Off) die Endschalterabfrage für die spezifizierte Achse. Wenn aktiviert, veranlasst es die entsprechenden Bits im Event Status Regis und Activity Status Register gesetzt zu werden, wenn es entweder auf den positiven oder negativen Endschalter trifft und die Achse wird sofort gestoppt. Wenn deaktiviert sind diese Bits nicht gesetzt, egal ob die Achsen auf einen Endschalter treffen oder nicht. GetLimitSwitchModeliest den Code für den aktuellen Status für die Endschalterabfrage.			
Einschränkung				
siehe	GetActivityStatus, GetEventStatus			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetLimitSwitchMode(hPosys#, axis, mode) GetLimitSwitchMode(hPosys#, axis)			

¤

Set/GetMotionCompleteMode

SetMotionCompleteMode
GetMotionCompleteMode

Syntax	SetMotionCompleteMode(hPosys, axis, mode) stdcall GetMotionCompleteMode(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	mode	commanded actual	0 1	
Definition	SetMotionCompleteMode bestimmt die Quelle für den Vergleich welches den Motion-C omplete Status der spezifizierten Achse bestimmt. Wenn auf "commanded" gesetzt, wird die Bewegung abgeschlossen betrachtet sobald die Sollgeschwindigkeit den Wert Null erreicht und keine weite Bewegung wird ausgeführt, ohne dass ein neues Kommando gesendet wird. Dieser Modus bleibt unbeeinflusst von der aktuellen Enkoderposition. Wenn im Modus "actual", dann wird das Motion-C omplete Bit gesetzt wenn die obige Bedingung "True" AND die aktuelle Enkoderposition befindet sich innerhalb des Settle Window (SetSettleWindow Kommando) für eine Anzahl Servoloops die mit dem Kommando SetSettleTi spezifiziert wurde. Der Settle "Timer" startet bei Null am Ende des Bahnprofils, so dass eine minimale Verzögerung von SettleTime Zyklen auftreten wird nach Beendigung des Bahnprofils. GetMotionCompleteMode liest den programmierten C ode für die aktuelle Einstellung.			
Einschränkung				
siehe	Set/GetSettleTime, Set/GetSettleWindow			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetMotionCompleteMode(hPosys#, axis, mode) GetMotionCompleteMode(hPosys#, axis)			

¤

Set/GetSampleTime

SetSampleTime
GetSampleTime

Syntax	SetSampleTime(hPosys, time) stdcall GetSampleTime(hPosys) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	POSYS® 18xx/19xx	Type		
	time	unsigned 32 bits	Range	51 to 220
			Scaling	unity
			Units	µsec/cycle
	POSYS® 7xx, 8xx, 8xx-B, 9xx	Type		
	time	unsigned 16 bits	Range	1 to 2^{15}-1
			Scaling	unity
			Units	µsec/cycle

Definition

SetSampleTimebestimmt die Updatrate für die POSYS®. Es bestimmt sowohl die Servoloop Updaterate als auch den Intervall für die Kalkulation neuer Bahnparameter für alle Servoachser Jedoch bestimmt es nicht die Kommutierungsrate für DC bürstenlose Servomotoren.

Dieser Wert wird in Mikrosekunden definiert.
Die POSYS® 18xx/19xx akzeptieren Werte in Abständen von 64 ~ 102.4 Mikrosekunden (abhär von der Anzahl Achsen); der übergebene Wert wird an den nächsten Vielfachen dieses Wertes gerundet.
Die POSYS® 7xx/8xx/8xx-B/9xx akzeptieren Werte in Abständen von 102,4 bzw. 153,6 Mikrosekunden; der übergebene Wert wird an den nächsten Vielfachen dieses Wertes gerundet.

GetSampleTimeliefert den programmierten Wert zurück.

Die minimale Zykluszeit wird von der Anzahl Achsen pro Karte bestimmt. Die folgende Tabelle z die möglichen Intervalle auf:

POSYS® 18xx und 19xx Serie

# aktivierte Achsen	Minimale Zykluszeit	Zykluszeit m/ Trace Capture	Zykluszeit pro Achse	Maximale Frequenz
1	102.4 µs	102.4 µs	102.4 µs	9.76 kHz
2	153.6 µs	153.6 µs	76.8 µs	6.51 kHz
3	204.8 µs	204.8 µs	68.3 µs	4.88 kHz
4	256 µs	256 µs	64 µs	3.91 kHz

POSYS® 7xx, 8xx, 8xx-B und 9xx Serie

# aktivierte Achsen	Minimale Zykluszeit bürsten-behaftete Servo und Schrittmotor Versionen	Minimale Zykluszeit bürstenlose Servo- und Mikroschritt-motor Versionen	Maximale Frequenz bürsten-behaftete Servo und Schrittmotor Versionen	Maximale Frequenz bürstenlose Servo- und Mikroschritt-motor Versionen
1	102.4 µs	153.6 µs	10 kHz	6.51 kHz
2	204.8 µs	307.2 µs	5 kHz	3.255 kHz
3	307.2 µs	460.8 µs	3.33 kHz	4.822 kHz
4	409.6 µs	614.4 µs	2.5 kHz	1.627 kHz

Einschränkung

Dieses Kommando betrifft die Zykluszeit aller Achsen und kann nicht individuell für jede Achse verwendet werden.

Dieses Kommando kann nicht verwendet werden, um kürzere als die erlaubten Zykluszeiten zu setzen. Bei dem Versuch wird der kleinst-mögliche Wert (siehe Tabelle) verwendet.

siehe

DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)
MotionScript® Aufruf-konvention	SetSampleTime(hPosys#, time) GetSampleTime(hPosys#)

¤

Set/GetSettleTime

SetSettleTime
GetSettleTime

Syntax	SetSettleTime(hPosys, axis, time) stdcall GetSettleTime(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Type		
	time	unsigned 16 bits	Range	0 to 2^{15}-1
			Scaling	unity
			Units	cycles
Definition	SetSettleTimedefiniert die Zeitspanne in Zyklen, nach der eine Achse sich innerhalb eines Positionsfensters befinden soll, ehe der axis-settled Indikator (im Activity Status Register) gesetzt wird. GetSettleTimeliest den programmierten Wert zurück für die spezifizierte Achse.			
Einschränkung				
siehe	Set/GetMotionCompleteMode, Set/GetSettleWindow, GetActivityStatus			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetSettleTime(hPosys#, axis, time) GetSettleTime(hPosys#, axis)			

¤

Set/GetSettleWindow

SetSettleWindow
GetSettleWindow

Syntax	SetSettleWindow(hPosys, axis, window) stdcall GetSettleWindow(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	POSYS® 18xx, 19xx	Type		
	window	unsigned 16 bits	Range	0 to 2^{16}-1
			Scaling	unity
			Units	counts
	POSYS® 7xx, 8xx, 8xx-B, 9xx	Type		
	window	unsigned 16 bits	Range	0 to 2^{15}-1
			Scaling	unity
			Units	counts
Definition	SetSettleWindowdefiniert die Fenstergröße innerhalb der sich eine spezifizierte Achse für eine mit dem Kommando SetSettleTime definierte Zeitspanne befinden muss, ehe der axis-settled Indikator (im Activity Status Register) gesetzt wird. GetSettleWindowliest den programmierten Wert zurück.			
Einschränkung				
siehe	Set/GetMotionCompleteMode, Set/GetSettleTime, GetActivityStatus			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetSettleWindow(hPosys#, axis, window) GetSettleWindow(hPosys#, axis)			

¤

Set/GetTrackingWindow

SetTrackingWindow
GetTrackingWindow

Syntax	SetTrackingWindow(hPosys, axis, window) stdcall GetTrackingWindow(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	POSYS® 18xx/19xx	Type		
	window	unsigned 16 bits	Range	0 to 2^{16}-1
			Scaling	unity
			Units	counts
	POSYS® 7xx, 8xx, 8xx-B, 9xx	Type		
	window	unsigned 16 bits	Range	0 to 2^{15}-1
			Scaling	unity
			Units	counts
Definition	SetTrackingWindowdefiniert Grenzen für die Istposition der spezifizierten Achse. Falls die Ach die spezifizierten Grenzen in die eine oder andere Richtung überschreitet, wird der Tracking Indikator (Bit 2 des Activity Status Registers) auf Null gesetzt. Wenn die Achse wieder in die gesetzten Grenzen zurückkehrt, wird der Tracking Indikator wieder auf 1 gesetzt. GetTrackingWindowliest den programmierten Wert für das aktuelle Tracking Window.			
Einschränkung				
siehe	GetActivityStatus, Set/GetActualPosition			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetTrackingWindow(hPosys#, axis, window) GetTrackingWindow(hPosys#, axis)			

¤

Status Register und AxisOut Indikator

GetActivityStatus

GetActivityStatus

Syntax	GetActivityStatus(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance		
	hPosys	CardHandle	Encoding Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
Ergebnis		Type		
	status	unsigned 16 bits	siehe unten	

Definition

GetActivityStatus liest den 16 Bit Activity Status Register der spezifizierten Achse. Jedes diese Bits liest den Register, welches kontinuierlich den Status der POSYS® anzeigt, ohne den Host z belasten. Es gibt keine direkte Möglichkeit den Status dieser Bits zu setzen oder zu löschen, da von der POSYS® direkt kontrolliert werden.

Die folgende Tabelle zeigt die Aufschlüsselung der Bedeutung der zurückgelieferten Daten für dieses Kommando.

Name	Bit Nummer	Beschreibung
Phasing Initialized	0	1 wenn die Phasen initialisiert sind (nur bürstenlose Servoversionen)
At maximum velocity	1	1 wenn die Bahn bei der maximalen Geschwindigkeit ist. Dieses Bit wird vom Regler bestimmt und nicht von der aktuellen Enkoderposition.
Tracking	2	1 wenn die Achse innerhalb des Tracking Windows ist.
Current profile mode	3-5	Enthält Bahnprofilinformationen wie folgt:

Bit 1	Bit 2	Bit 3	Profile Mode
0	0	0	Trapezoidal
0	0	1	Velocity contouring
0	1	0	S-curve
0	1	1	Electronic gear
1	0	0	External profile mode

reserved	6	nicht verwendet; kann 0 oder 1 sein
Axis settled	7	1 wenn die Achse ruht
Motor on/off	8	1 wenn Motor Mode ist ON, 0 wenn OFF.
Position capture	9	Latch Eingang erfasst, aber noch nicht gelesen wurde. Das Kommando GetCaptureValue muss ausgeführt werden ehe eine neue Erfassung möglich ist.
In-motion	10	1 wenn auf der Achse vom Regler eine Bewegung ausgeführt wird.
In positive limit	11	1 wenn der positive Endschaltereingang

			akiviert wurde.
	In negative limit	12	1 wenn der negative Endschaltereingang akiviert wurde.
	Profile segment	13-15	Nur im S-Kurven Profilmodus. Wert ist 0 wenn keine Bewegung ausgeführt wird. Ansonsten Werte 1-7 wenn in Bewegung und zeigt an in welchem Segment es sich gerade befindet.

Einschränkung	
siehe	GetEventStatus, GetSignalStatus, ResetEventStatus
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)
MotionScript® Aufruf-konvention	GetActivityStatus(hPosys#, axis)

¤

GetAxis_In_Out_Home

Syntax	GetAxis_In_Out_Home(hPosys, statID) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	statID	statusID	02	
Definition	GetAxis_In_Out_Home liefert den Status der 4 Achsen-spezifischen AxisIn-Eingänge, AxisOut-Ausgänge und der Home-Eingänge. Die 4 niederwertigsten Bits 0-3 beziehen sich auf d Status der Home-Eingänge, die Bits 4-7 auf die AxisIn-Eingänge und die Bits 8-11 auf die AxisOut-Ausgänge. Die exakte Bedeutung ist wie folgt: Bits 0-3: Home inputs axis 0-3 (X-W) Bits 4-7: AxisIn inputs axis 0-3 (X-W) Bits 8-11: AxisOut outputs axis 0-3 (X-W) Bit 12: Reset cause - Software Bit 13: Reset cause - Undervoltage Bit 14: Reset cause - External Bit 15: Reset cause - Watchdog			
Einschränkung	Diese Ausgänge werden nicht von einem HardReset, SoftReset oder externem Reset betroffen. Sie werden ihren ursprünglichen Status beibehalten.			
siehe				
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	GetAxis_In_Out_Home(hPosys#, statID)			

¤

GetDriveStatus

nur gültig mit POSYS® 1800/1900 falls mit Motion Control Chipsatz bestückt > V2.x

Syntax	GetDriveStatus(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
Ergebnis	Name	Type		
	status	unsigned 16 bits	see below	
Definition	GetDriveStatusliest den Drive Status Register für die spezifizierte Achse. Alle Bits in diesem Statuswort werden vom Motion Controller gesetzt und zurückgesetzt.			
	Name	Bit	Beschreibung	
	Reserved	0-3		
	In Holding	4	1 falls sich die Achse in einem Holding Current Status befindet. Die Stromausgabe ist begrenzt durch die Strombegrenzung.	
	Reserved	5-15		
Einschränkung				
siehe				
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	GetDriveStatus(hPosys#, axis)			

¤

GetEventStatus

Syntax	GetEventStatus(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
Ergebnis	Name	Type	Encoding	
	status	unsigned 16 bits	siehe unten	

Definition

GetEventStatus liest das Event Status Register der spezifizierten Achse.
Die folgende Tabelle listet die Aufschlüsselung der zurückgelieferten Werte für dieses Kommando.

Name	Bit(s)	Beschreibung
Motion Complete	0	1 wenn Bewegung abgeschlossen. SetMotionCompleteMode bestimmt ob dieser Wert auf der EnkoderSollPosition oder EnkoderIstPosition basiert.
Wrap-around	1	1 wenn die aktuelle Enkoderposition überläuft vom Maximalwert zum Minimalwert und umgekehrt
Breakpoint 1	2	1 wenn Breakpoint 1 getriggert wurde
Capture received	3	1 wenn eine Positionserfassung erfolgte
Motion error	4	1 wenn ein Motion Error auftritt
In positive limit	5	1 wenn die Achse den positiven Endschalter anfährt
In negative limit	6	1 wenn die Achse den negativen Endschalter anfährt
Instruction error	7	1 wenn Kommandofehler auftritt
reserved	8-10	nicht verwendet, kann 0 oder 1 sein
Commutation error	11	1 wenn ein Kommutierungsfehler vorkommt
reserved	12-13	nicht verwendet, kann 0 oder 1 sein
Breakpoint 2	14	1 wenn Breakpoint 2 getriggert wurde
reserved	15	nicht verwendet, kann 0 oder 1 sein

Einschränkung	Alle dieser Bits in diesem Statuswort werden von der POSYS® gesetzt und vom Host gelöscht. Um diese Bits zu löschen, verwenden Sie das Kommando ResetEventStatus.
siehe	GetActivityStatus, GetSignalStatus, ResetEventStatus, GetDriveStatus
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)
MotionScript® Aufruf-konvention	GetEventStatus(hPosys#, axis)

¤

GetSignalStatus

Syntax	GetSignalStatus(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
Ergebnis	Name	Type		
	siehe unten	unsigned 16 bits		

Definition

GetSignalStatus liefert den Inhalt des Signal Status Registers für die spezifizierte Achse. Der Signal Status Register beinhaltet den aktuellen Wert für verschiedene Hardwaresignale, die mit jeder Achse der POSYS® verbunden sind. Der zurückgelesene Wert wird mit dem Signal Sense Register kombiniert (SetSignalSense Kommando) und wird dann dem Anwender zurückgeliefert. Für jedes Bit im Signal Sense Register welches auf 1 steht, wird das entsprechende Bit im GetSignalStatus-Kommando invertiert, so dass ein "low"-Signal als 1 und ein "high"-Signal als 0 interpretiert wird. Umgekehrt, jedes Bit, daß im Signal Sense Register 0 ist, ist der entsprechende Bit im GetSignalStatus-Kommando nicht invertiert, sodass ein "low"-Signal als 0 und ein "high"-Signal als 1 interpretiert wird.

Alle Bits im GetSignalStatus-Kommando sind Eingänge mit Ausnahme von AxisOut. Der Wert der für dieses Bit gelesen wird, ist gleich dem aktuellen Wert der ausgegeben wird vom AxisOut-Mechanismus. Siehe SetAxisOutSource-Kommando für Details.

Name	Beschreibung	Bit Number
status	Encoder A	0
	Encoder B	1
	Encoder Index	2
	Encoder Home	3
	Positive limit	4
	Negative limit	5
	AxisIn	6
	Hall A	7
	Hall B	8
	Hall C	9
	AxisOut	10
	reserved	11-15

Einschränkung	
siehe	GetActivityStatus, GetEventStatus, ResetEventStatus, Set/GetSignalSense
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)
MotionScript® Aufruf-konvention	GetSignalStatus(hPosys#, axis)

¤

ResetEventStatus

Syntax	ResetEventStatus(hPosys, axis, mask) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
			POSYS® 700, 800, 800B all versions & POSYS® 1800, 1900 Firmware <=1.5	POSYS® 1800/1900 Firmware >2.x
	mask	Motion Complete Wrap-Around Breakpoint 1 Capture Received Motion Error In positive Limit In negative Limit Instruction Error Commutation Error Breakpoint 2	0001h 0002h 0004h 0008h 0010h 0020h 0040h 0080h 0800h 4000h	FFFEh FFFDh FFFBh FFF7h FFEFh FFDFh FFBFh FF7Fh F7FFh BFFFh
Definition	ResetEventStatus löscht (setzt zurück auf 0) jedes Bit im Event Status Register für die spezifizierte Achse, welches eine 0 aufweist in der mask, der mit diesem Kommando geschickt wird. Alle anderen Bits im Event Status Register (Bits mit einer 1) bleiben unberücksichtigt. POSYS 18001900 mit Firmware >2.x Ereignisse die eine Änderung im Operation Mode oder Trajectory Generation erfordern im generellen, dass das korrespondierende Bit im Event Status Register gelöscht wird ehe wieder zum normalen Betrieb zurück gekehrt wird. Das ist z.B. ehe Operating Mode wiederhergestellt wird (in den Fällen in denen ein Ereignis Einfluss auf den Operating Mode hatte) oder ehe ein neuer Fahrbefehl ausgegeben wird (in den Fällen in denen ein Ereignis Einfluss auf den Trajectory Loop hatte). Die Ausnahme ist *Motion Error*. Hier muss das Bit nicht gelöscht werden, falls das auslösende Ereignis die Deaktivierung der Position Loop zur Folge hatte.			
Einschränkung				
siehe	GetEventStatus			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	ResetEventStatus(hPosys#, axis, mask)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

SetAxisOutMask
GetAxisOutMask

nur gültig mit POSYS® 1800/1900 falls mit Motion Control Chipsatz bestückt > V2.x

Syntax	SetAxisOutMask(hPosys, axis, sourceAxis, sourceRegister, selectionMask, senseMask) stdcall GetAxisOutMask(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	sourceAxis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	sourceRegister	disabled EventStatus ActivityStatus SignalStatus DriveStatus	0 1 2 3 4	
	selectionMask	see below	bitmask	
	senseMask	see below	bitmaks	

Definition

SetAxisOutMask definiert welche Ereignisse das AxisOut Pin von *axis* schaltet. Die *sourceRegister* und *sourceAxis* Argumente spzifizieren welche Register von welcher Achse verwendet werden um den AxisOut-Ausgang zu schalten. Für jedes Bit in *selectionMask* welches auf 1 gesetzt wurde, wird das korrespondierende Bit vom spezifizierten *sourceRegister* ausgewählt um AxisOut auf *active* zu schalten. Die *senseMask* definiert welchen Status ein Bit haben muss ur als *active* zu gelten. Eine 1 in der *senseMask* bedeutet, dass eine 1 im korrespondierenden Bit AxisOut aktiv schaltet. Falls mehrere Bits in *sourceRegister* ausgewählt wurden, wird AxisOut akti sobald eines der Bits, abhängig von Sense diesen Zustand erfordert.

GetAxisOutMask liest das Mapping des AxisOut-Ausganges der spezifizierten Achse.

Die nachfolgende Tabelle zeigt die Quelle der Kombinationen von Bit und Register.

Encoding of bit	Register = event status	Register = activity status	Register = signal status	Drive Status Register
0	Motion Complete	Phasing initialized	Encoder A	
1	Wrap-around	At maximum velocity	Encoder B	
2	Breakpoint 1	Tracking	Encoder index	
3	Position capture		Home	
4	Motion error		Positive limit	In Holding
5	In positive limit		Negative limit	
6	In negative limit		AxisIn	
7	Instruction error	Axis settled	Hall sensor 1	
8	Disable	Motor on/off	Hall sensor 2	

	9		Position capture	Hall sensor 3	
	0Ah		In motion		
	0Bh	Commutation error	In positive limit		
	0Ch		In negative limit		
	0Dh				
	0Eh	Breakpoint 2			
	0Fh				

siehe	Set/GetSignalSense
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)
MotionScript® Aufruf-konvention	SetAxisOutMask(hPosys#, axis, sourceAxis, sourceRegister, selectionMask, senseMask) GetAxisOutMask(hPosys#, axis)

¤

Set/GetAxisOutSource

SetAxisOutSource
GetAxisOutSource

Syntax	SetAxisOutSource(hPosys, axis, sourceAxis, bit, register) stdcall GetAxisOutSource(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	sourceAxis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	bit	siehe unten	0 to 15	
	register	disabled EventStatus ActivityStatus SignalStatus	0 1 2 3	

Definition

SetAxisOutSource mapped das spezifizierte Bit des spezifizierten Status Register von Achse n zum AxisOut-Ausgang der spezifizierten Achse. Der Zustand des AxisOut-Ausganges wird danac den Zustand von “bit” überwachen. Falls das Register abwesend ist (Zustand 0), wird “bit” ignor und der spezifizierte AxisOut-Ausgang ausgeschaltet (inaktiv).

GetAxisOutSource liest das Mapping des AxisOut-Ausganges der spezifizierten Achse.

Die nachfolgende Tabelle zeigt die Quelle der Kombinationen von Bit und Register.

Encoding of bit	Register = event status	Register = activity status	Register = signal status
0	Motion Complete	Phasing initialized	Encoder A
1	Wrap-around	At maximum velocity	Encoder B
2	Breakpoint 1	Tracking	Encoder index
3	Position capture		Home
4	Motion error		Positive limit
5	In positive limit		Negative limit
6	In negative limit		AxisIn
7	Instruction error	Axis settled	Hall sensor 1
8		Motor on/off	Hall sensor 2
9		Position capture	Hall sensor 3
0Ah		In motion	
0Bh	Commutation error	In positive limit	
0Ch		In negative limit	
0Dh			
0Eh	Breakpoint 2		
0Fh			

siehe: Set/GetSignalSense

DLL: Delphi, C++, VB: wir verweisen auf die verschiedenen Importdateien (imp*.*)

MotionScript® Aufruf-: SetAxisOutSource(hPosys#, axis, sourceAxis, bit, register)
GetAxisOutSource(hPosys#, axis)

konvention

¤

Set/GetSignalSense

SetSignalSense
GetSignalSense

Syntax	SetSignalSense(hPosys, axis, mask) stdcall GetSignalSense(hPosys, axis) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
		Indicator	Encoding	Bit number
	mask	Encoder A Encoder B Encoder Index Encoder Home Positive Limit Negative Limit AxisIn Hall A Hall B Hall C AxisOut StepOutput MotorDirection reserved	0001h 0002h 0004h 0008h 0010h 0020h 0040h 0080h 0100h 0200h 0400h 0800h 1000h	0 1 2 3 4 5 6 7 8 9 10 11 12 13-15
Definition	SetSignalSensedefiniert den "Sense" der Signale, die mit dem Signal Sense Register verbunde sind, dabei eine Bitmaske verwendend die den Bits des Signal Status Registers entsprechen, fü spezifizierte Achse. Jedes "Sense"-Bit welches = 0, ist der Eingang „active low“, oder nicht invertiert. Jedes "Sense"-Bit welches = 1, ist der Eingang „active high“, oder invertiert. GetSignalSense liest die aktuelle "Sense"-Maske. Für Enkoderindex/Home: Falls das SenseBit = 1, dann wird eine Erfassung erfolgen bei einer low-to-high Signaländerung. Andernfalls wird die Erfassung bei einer high-to-low Signaländerung erfolgen. Für die positiven und negativen Endschaltereingänge: Falls das SenseBit = 1, dann erfolgt ein Endschalterereignis wenn das Signal "high" ist. Andernfalls wird die Erfassung erfolgen wenn das Signal "low" ist. Das AxisOut Signal wird invertiert wenn der SenseBit auf 1 gesetzt wurde; andernfalls ist es nic invertiert. Wenn das TaktsignalBit = 1, wird jeder Schritt der erfolgen soll auf Basis einer low-to-high Flan erfolgen. Andernfalls wird ein Schritt bei einer high-to-low Flanke erfolgen. Das Setzen des RichtungsSignalBits hat zur Folge, dass die Richtung umgekehrt wird. GetSignalSense liefert den gegenwärtigen prgrammierten Status der SignalSenseMask zurück.			
Einschränkung	Das Invertieren der Enkoder Signale A, B oder I kann zur Folge haben, dass der Capturemechanismus nicht mehr richtig funktioniert. Bitte kontakten Sie uns wenn Sie Unterstützung benötigen.			
siehe	GetSignalStatus			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetSignalSense(hPosys#, axis, mask) GetSignalSense(hPosys#, axis)			

¤

Traces

GetTraceCount

GetTraceCount

Syntax	GetTraceCount(hPosys) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
Ergebnis	Value	Type		
	trace count	unsigned 32 bits	Range	0 to 2^{32}-1
			Scaling	unity
			Units	samples
Definition	GetTraceCount liefert die Anzahl erfasste Datenfelder (Variablenwerte) die im Tracebuffer gespeichert wurden, seitdem der Trace begonnen hat.			
Einschränkung	Falls der Trace Modus auf „rolling“ eingestellt ist und der Buffer umbricht (wrap-around), wird GetTraceCount die Anzahl erfasste Daten im gefüllten Buffer zurückliefern.			
siehe	ReadBuffer, Set/GetTraceStart, Set/GetTraceStop, Set/GetBufferLength			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	GetTraceCount(hPosys#)			

¤

GetTraceStatus

Syntax	GetTraceStatus(hPosys) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
Ergebnis	Name	Type		
	siehe unten	unsigned 16 bits		
Definition	GetTraceStatus liefert den aktuellen Tracestatus zurück. Die Definition der einzelnen Statusbits ist wie folgt:			
	Bit Number	Name	Definition	
	0	Mode	0 wenn die Tracefunktion in Einmalmodus (one-time), 1 wenn in kontinuierlichem (rolling) Modus.	
	1	Activity	1 wenn Tracefunktion aktiv (Datenerfassung läuft gerade), 0 falls nicht	
	2	DataWrap	1 wenn der Trace umbricht (wrap-around), 0 falls nicht. Falls 0 dann wurde der Buffer noch nicht gefüllt und alle erfassten Daten sind intakt. Falls 1, dann hat es einen Umbruch gegeben und die Datenerfassung beginnt den Buffer von vorn wieder zu füllen; vorher erfasste Daten können überschrieben worden sein, falls nicht explizit schon vorher vom Host mit dem Kommando ReadBuffer abgerufen, während der Trace aktiv ist.	
Einschränkung				
siehe	Set/GetTraceStart, Set/GetTraceMode			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	GetTraceStatus(hPosys#)			

¤

Set/GetTraceMode

SetTraceMode
GetTraceMode

Syntax	SetTraceMode(hPosys, mode) stdcall GetTraceMode(hPosys) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	mode	OneTime RollingBuffer	0 1	
Definition	SetTraceModekonfiguriert den Buffer für die nächste Erfassung. Im "OneTime"-Modus wird die Erfassung fortlaufen, bis der Speicher gefüllt wurde und dann stoppen. Im "Rolling"-Modus wird Erfassung bis zum Ende des Buffers fortlaufen und fängt dann wieder vom Anfang des Speicher die alten Daten mit neuen Daten zu überschreiben. GetTraceModeliest den programmierten Modus zurück.			
Einschränkung				
siehe	GetTraceStatus			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetTraceMode(hPosys#, mode) GetTraceMode(hPosys#)			

¤

Set/GetTracePeriod

SetTracePeriod
GetTracePeriod

Syntax	SetTracePeriod(hPosys, period) stdcall GetTracePeriod(hPosys) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	POSYS® 18xx/19xx	Type		
	period	unsigned 16 bits	Range	1 to 2^{16}-1
			Scaling	unity
			Units	cycles
	POSYS® 7xx, 8xx, 8xx-B, 9xx	Type		
	period	unsigned 16 bit	Range	1 to 2^{15}-1
			Scaling	unity
			Units	cycles
Definition	SetTracePeriod definiert die Zeitspanne (in Anzahl Zyklen) zwischen aufeinander folgenden Tracepunkten. Da heisst um z.B. über einen längeren Zeitraum eine Erfassung laufen zu lassen, muss nicht mi jeder Updaterate der neue Wert gespeichert werden, sondern es kann jeder 4., 6. oder 15. Wer gespeichert werden. GetTracePeriod liefert den programmierten Wert zurück.			
Einschränkung				
siehe	Set/GetSampleTime, Set/GetTraceStart, Set/GetTraceStop			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetTracePeriod(hPosys#, period) GetTracePeriod(hPosys#)			

¤

Set/GetTraceStart

SetTraceStart
GetTraceStart

Syntax	SetTraceStart(hPosys, triggerAxis, condition, triggerBit, triggerState) stdcall GetTraceStart(hPosys) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	triggerAxis	Axis1 Axis2 Axis3 Axis4		0 1 2 3
	condition	Immediate Next update Event Status register bit Activity Status register bit Signal Status register bit		0 1 2 3 4
	triggerBit	Status register bit		0 to 15
	triggerState	Triggering state of the bit		0 (value = 0) 1 (value = 1)

Definition

SetTraceStartsetzt die Bedingung für den Beginn der Erfassung. Die "immediate"-Bedingung erfordert keine Achsen-Spezifikation und wird sofort ausgeführt. Die anderen 4 Bedingungen benötigen die Spezifikation einer Achse. Sobald die Bedingung eintritt, wird die Erfassung starte Wenn als Trigger ein Status Register Bit ausgewählt wurde, müssen die Bitnummer und der Sta im Argument angegeben werden. Die Erfassung beginnt, wenn der Bit den geforderten Zustand erhält (0 oder 1).
Sobald die Erfassung gestartet wurde, wird der Trace-Start Indikator resettet und das SetTraceStart Kommando muss erneut ausgeführt werden, ehe eine neue Erfassung beginnen kann.

GetTraceStartliest den programmierten Zustand zurück

Beispiel:
Falls es gewünscht wird, dass die nächste Erfassung mit einem Update-Kommando für die 3. Ac beginnt, dann wird "1" für die Bedingung gesetzt, eine "2" für die Achse und Bitnummer und Sta können mit Nullen gefüllt werden, da sie nicht benötigt werden.

Falls es gewünscht wird, dass die Erfassung beginnt, wenn Bit 7 im Activity Status Register für Achse 2 auf Null geht, dann wird die Erfassung auf folgende Weise geladen: Eine "3" wird gelad für die Bedingung, eine "1" für die Achse, eine "7" für die Bitnummer und eine "0" für den Zusta (status).

Encoding of bit	Register = event status	Register = activity status	Register = signal status
0	Motion Complete	Phasing initialized	Encoder A
1	Wrap-around	At maximum velocity	Encoder B
2	Breakpoint 1	Tracking	Encoder index
3	Position capture		Home
4	Motion error		Positive limit
5	In positive limit		Negative limit
6	In negative limit		AxisIn
7	Instruction error	Axis settled	Hall sensor 1
8		Motor on/off	Hall sensor 2
9		Position capture	Hall sensor 3
0Ah		In motion	
0Bh	Commutation error	In positive limit	
0Ch		In negative limit	

	0Dh	
	0Eh	Breakpoint 2
	0Fh	

Einschränkung

siehe Set/GetBufferLength, GetTraceCount, Set/GetTraceMode, Set/GetTracePeriod, Set/GetTraceStop

DLL: Delphi, C++, VB wir verweisen auf die verschiedenen Importdateien (imp*.*)

MotionScript® Aufruf-konvention SetTraceStart(hPosys#, triggerAxis, condition, triggerBit, triggerState)
GetTraceStart(hPosys#)

¤

Set/GetTraceStop

SetTraceStop
GetTraceStop

Syntax	SetTraceStop(hPosys, triggerAxis, condition, triggerBit, triggerState) stdcall GetTraceStop(hPosys) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	triggerAxis	Axis1 Axis2 Axis3 Axis4		0 1 2 3
	condition	Immediate Next update Event Status register bit Activity Status register bit Signal Status register bit		0 1 2 3 4
	triggerBit	Status register bit		0 to 15
	triggerState	Triggering state of the bit		0 (value = 0) 1 (value = 1)

Definition

SetTraceStopsetzt die Bedingung fest für das Stoppen einer Erfassung. Die "immediate"-Bedingung erfordert keine Achsen-Spezifikation und wird sofort ausgeführt und die Erfassung gestoppt. Die anderen 4 Bedingungen benötigen die Spezifikation einer Achse. Sobald die Bedingung eintritt wird die Erfassung gestoppt.
Wenn als Trigger ein Status Register Bit ausgewählt wurde, müssen die Bitnummer und der Sta im Argument angegeben werden. Die Erfassung stoppt wenn der Bit den geforderten Zustand erhält (0 oder 1).
Sobald eine Erfassung beendet wurde, wird der "trace-stop"-Indikator resettet und das SetTraceStop Kommando muss erneut ausgeführt werden, ehe eine neue Erfassung beendet werden kann.

GetTraceStopliest den programmierten Wert zurück für die Stopp-Bedingung

Beispiele:
Falls es gewünscht wird, dass die Erfassung beim nächsten Update-Kommando für Achse "3" beendet wird, dann muss als Bedingung eine "1" und eine "2" für die Achse eingegeben werden Bitnummer und Status können mit Nullen geladen werden, da sie nicht benötigt werden.
Falls es gewünscht wird, dass die Erfassung stoppt, wenn Bit 7 im Activity Status Register für Achse 2 auf Null geht, dann wird die Beendigung der Erfassung auf folgende Weise geladen: Ein "3" wird geladen für die Bedingung, eine "1" für die Achse, eine "7" für die Bitnummer und eine für den Zustand (status).

Encoding of bit	Register = event status	Register = activity status	Register = signal status
0	Motion Complete	Phasing initialized	Encoder A
1	Wrap-around	At maximum velocity	Encoder B
2	Breakpoint 1	Tracking	Encoder index
3	Position capture		Home
4	Motion error		Positive limit
5	In positive limit		Negative limit
6	In negative limit		AxisIn
7	Instruction error	Axis settled	Hall sensor 1
8		Motor on/off	Hall sensor 2
9		Position capture	Hall sensor 3
0Ah		In motion	
0Bh	Commutation error	In positive limit	
0Ch		In negative limit	

	0Dh	
	0Eh	Breakpoint 2
	0Fh	

Einschränkung

siehe	GetTraceCount, Set/GetTraceStart, GetTraceStatus
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)
MotionScript® Aufruf-konvention	SetTraceStop(hPosys#, triggerAxis, condition, triggerBit, triggerState) GetTraceStop(hPosys#)

¤

© POSYS Motion Control GmbH & Co.KG, 2013

SetTraceVariable
GetTraceVariable

manche Optionen sind Firmware-abhängig für POSYS® 1800/1900 falls mit Motion Control Chipsatz bestückt >< V2.x

Syntax	SetTraceVariable(hPosys, variableNumber, traceAxis, variable) stdcall GetTraceVariable(hPosys, variableNumber) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente Ergebnis	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	variableNumber	Variable1		0
		Variable2		1
		Variable3		2
		Variable4		3
	axis	Axis1		0
		Axis2		1
		Axis3		2
		Axis4		3
	variable	None (disable variable)		0
		Position Error		1
		Commanded Position		2
		Commanded Velocity		3
		Commanded Acceleration		4
		Actual Position		5
		Actual Velocity		6
		Motor Command		7
		Chip Time		8
		Capture Register		9
		Position Loop Integral Sum		Ah
		Position Loop Derivative		Bh
		Event Status Register		Ch
		Activity Status Register		Dh
		Signal Status Register		Eh
		Phase Angle		Fh
		Phase Offset		10h
		Phase A		11h
		Phase B		12h
		Phase C		13h
		Analog Input 1		14h
		Analog Input 2		15h
		Analog Input 3		16h
		Analog Input 4		17h
		Analog Input 5		18h
		Analog Input 6		19h
		Analog Input 7		1Ah
		Analog Input 8		1Bh
		PID Servo Error		1Ch
		Phase Angle Scaled *		1Dh
		Drive Status Register *		38h
		Position Loop Integral Contribution *		39h
		PID Output (Biquad1 Input) *		40h
		Biquad1 Output (Biquad2 Input) *		41h

Definition

SetTraceVariable weist eine Variable einer spezifischen VariableNumber-Position im TraceBuffe (Zwischenspeicher) zu. Die Variable wird immer eine 32-Bit große Buffergröße beanspruchen. 16-Bit Werte werden auf vorzeichenabhängige 32-Bit Werte erweitert. Bis zu 4 Variablen könne auf einmal für eine Erfassung ausgewählt werden. Alle Arten von Achsen- und Erfassungsvariablenvariationen werden unterstützt.

Alle Variablenzuweisungen müssen kontinuierlich sein und fangen an mit der variableNumber =

GetTraceVariable liefert die Variable und die Achse zurück für die spezifizierte variableNumber

Beipsiel: Um eine Erfassung von 3 Variablen zu definieren, Sollbeschleunigung für Achse 1, Istposition für Achse 1 und das Event Status Word für Achse 3, muss folgende Sequenz

	programmiert werden. Zuerst wird ein SetTraceVariable Kommando mit traceId von "0", Achse und eine Variable "4" übergeben, dann ein SetTraceVariable Kommando mit traceId "1", Achse und eine Variable "5" übergeben. Schließlich wird ein SetTraceVariable Kommando mit traceId " Achse "2" und eine Variable "0C h" übergeben.
Einschränkung	Wenn ActualVelocity als Tracevariable ausgewählt wird, ist der zurückgemeldete Wert die absolute Änderung in der Position zwischen den Tracesamples und nicht die Änderung pro Zyklus. Auf diese Art kann die Traceperiode verwendet werden um die Sampleperiode für Geschwindigkeit zu ändern. * mit Asterisk gekennzeichnete Optionen unter *variable* sind nur verfügbar in der Firmware Version >2.x für POSYS® 1800/1900.
siehe	Set/GetTracePeriod, Set/GetTraceStart, Set/GetTraceStop
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)
MotionScript® Aufruf-konvention	SetTraceVariable(hPosys#, variableNumber, traceAxis, variable) GetTraceVariable(hPosys#, variableNumber)

¤

Verschiedenes

GetChecksum

GetChecksum

Syntax	GetChecksum(hPosys) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
Ergebnis		Type		
	checksum	unsigned 32 bits		
Definition	GetChecksum liest den internen 32-Bit Checksummenwert. Der zurückgelieferte Wert hängt von der Firmwareversion ab. Für weitere Informationen kontaktieren Sie bitte POSYS Motion Control GmbH & Co.KG.			
Einschränkung				
siehe				
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	GetChecksum(hPosys#)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

GetHostIOError
GetInstructionError (für POSYS® 1800/1900 mit Firmware > V2.0)

Syntax	GetHostIOError(hPosys) stdcall GetInstructionError(hPosys) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
Ergebnis	Name	Type	Range	
	error code	unsigned 16 bit	0 - Eh	

Definition

POSYS® 7xx, 8xx, 8xx-B, 9xx

GetHostIOError / GetInstructionError liefert den Code des letzten Host I/O Fehlers, resetted dann beide auf Null sowohl den Fehler als auch das Host I/O Bit im Status-Read Wort. Gewöhnli wird dieses Kommando nur dann verwendet, nachdem der Host I/O Error Bit im Status-Read W anzeigt, dass ein I/O Fehler aufgetreten ist.

POSYS® 18xx, 19xx

GetHostIOError / GetInstructionError liefert den Code des letzten Host I/O Fehlers, resetted dann beide auf Null sowohl den Fehler als auch das Host I/O Bit im Status-Read Wort. Gewöhnli wird dieses Kommando nur dann verwendet, nachdem der Host I/O Error Bit im Status-Read W anzeigt, dass ein I/O Fehler aufgetreten ist.

GetHostIOError / GetInstructionError löscht den Host I/O Error Bit im ParallelStatusRegister n noch falls es über das parallele Interface (ISA-/PC I-Bus) aufgerufen wurde. Vor diesem Wechs hat ein Aufruf von GetHostIOError über jedwede Schnittstelle erfolgen können.

Error Code	Encoding
No error	0
Processor reset	1
Invalid instruction	2
Invalid axis	3
Invalid parameter	4
Trace running	5
reserved	6
Block out of bounds	7
Trace buffer zero	8
Bad serial checksum	9
reserved	Ah
Invalid negative value	Bh
Invalid parameter change	Ch
Invalid move after limit condition / event triggered stop	Dh
Invalid move into limit	Eh
Invalid Operating Mode restore after event-triggered change	10h (nur POSYS 1800/1900 >2.x)
Invalid Operating Mode for command	11h (nur POSYS 1800/1900 >2.x)

Einschränkung	
siehe	GetEventStatus, ResetEventStatus
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)
MotionScript® Aufruf-konvention	GetHostIOError(hPosys#) GetInstructionError(hPosys#)

¤

GetProcessorStatus

Syntax	GetProcessorStatus(hPosys) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
Ergebnis	liefert ein 16-Bit Wort. Bit 0-12: unbenutzt, gesetzt zu 0 Bit 13: Holds value of HostIOError signal, 1 falls ein HostIOError aufgetreten ist Bit 14: Holds value of HostIntrpt signal. Eine 1 zeigt an das Signal ist "high". Bit 15: Holds value of HostRdy signal. Eine 1 zeigt an der Level ist "high".			
Definition	GetProcessorStatus kann zu jeder Zeit aufgerufen werden. Die Statusabfrage liefert Informationen über die Interfacesignale des Motion Control Chipsatzes.			
Einschränkung				
siehe				
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	GetProcessorStatus(hPosys#)			

¤

GetVersion

Syntax	GetVersion(hPosys) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
Ergebnis	Name		Type	
	version info high-word		unsigned 16 bits	
	version info low-word		unsigned 16 bits	
Definition	GetVersion returns product information encoded as shown above.			
	Product information	Definition		Encoding
	Produkt Familie Motortyp	POSYS®		2 oder 5 (Navigator oder Magellan)
		bürstenbehaftete Servo		1
		bürstenlose Servo		3
		Mikroschritt		4
		Takt/Richtung		5
		alle Motortypen (nur Magellan)		8
	Anzahl Chips			1, 2
	Anzahl unterstützte Achsen			1, 2, 3, 4
	spezielle Attribute	reserviert		0 bis 3
	kundenspezifischer Code	reserviert		0 bis 255
	Hauptsoftware-version			0 bis 15
	Nebensoftware-version			0 bis 15
Einschränkung	Diese Nummer liefert keine Information darüber ob eine PCI-, ISA- oder PC/104- oder andersformatige (z.B. Kundenspezifische) Karte eingesetzt wird zurück, sondern nur die interne Funktionsinformation des eingesetzten Motion Control Chipsatzes. Für Informationen bzgl. der eingesetzten Karte verwenden Sie das Kommando ReadPLD.			
siehe				
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	GetVersion(hPosys#)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

HardReset
MCReset

Syntax	HardReset(hPosys) stdcall alternatively MCReset(hPosys) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	

Definition

HardResetführt einen Hardwarereset auf der Positioniersteuerkarte an der mit cardAddress spezifizierten Adresse aus. Der interne Motion C ontrol Prozessor wird resettet und die optische Schalter für den ±10V Ausgang werden ausgeschaltet, d.h. die Ausgänge werden auf Ground gelegt. Alle Aus- und Eingänge werden in ihren Power-On Status zurückgesetzt. HardResethat den gleichen Effekt wie wenn der externe Reset auf "low" geht.

Um die Positioniersteuerkarte neu zu initialisieren, ist es notwendig eine SoftResetauszuführen, gefolgt von einem erneuten Laden der Filterparameter und weiterer notwendiger kundenspezifischer Einstellungen, als auch das Kommando DAC OutputsOn, um die optischen Schalter für den analogen Spannungsausgang für die ±10V Drehmomentsollwertvorgabe zu aktivieren. Nur die analogen Ausgänge sind mit optischen Schaltern verbunden. Die PWM-Ausgä verwenden andere Leitungen und sind nicht mit den optischen Schaltern verbunden.

MCResetführt einen HardResetder Motion Control Karte aus ohne Reload des FPGAs. Der Rese wird schneller ausgeführt. Ansonsten gleicht MC Reset dem Kommando HardReset.

Anmerkung: POSYS® 700 und 800
Mit Ausnahme der AxisOut-Ausgänge werden die anderen Ausgänge nicht automatisch resettet (kundenspezifische Ausgänge und Verstärkerfreigaben). Sollen diese Ausgänge ebenso zurückgesetzt werden, ist es notwendig noch das Kommando WriteIO(ioAddress,outputStatus) auszuführen. HardResethat den gleichen Effekt wie wenn der externe Reset auf "low" geht.

Acceleration	0	Kp	0
ActualPosition	0	Kvff	0
Active Operating Mode*	0033h *	LimitSwitchMode	1
		MotionCompleteMode	0
ActualPositionUnits *	0 or motor dependent*	Motion Error Event Action *	motor dependent *
AutoStopMode	1		
Auxiliary Encoder Source *	0 *	MotorBias	0
		MotorCommand	0
AxisMode	1	MotorLimit	32767
AxisOutSource	0	MotorMode *	1 or motor dependent*
AxisOut Register *	0 *	Operating Mode *	0033h *
AxisOut Selection Mask *	0 *	PhaseAngle	0
		Phase Correction Mode	motor dependent
AxisOut Sense Mask *	0 *	PhaseCounts	motor dependent
BiQuadCoefficients *	0	PhaseInitializeMode	0
Breakpoint 1	0	PhaseInitializeTime	0
Breakpoint 2	0	PhaseOffset *	65353 or -1 *
BreakpointValue 1	0	PhasePrescale	0
BreakpointValue 2	0	Position	0
Breakpoint Update Mask 1 *	0 *	PositionErrorLimit	siehe notes
		Positive Limit Event Action *	8 *
Breakpoint Update Mask 2 *	0 *		
		Negative Limit Event Action *	8 *
BufferLength	0		
BufferReadIndex	0	ProfileMode	0
BufferStart	siehe notes	PWM Frequency *	motor dependent *
BufferWriteIndex	0	SampleTime	siehe notes
CaptureSource	0	SettleTime	0
Commutation Mode *	siehe notes	SettleWindow	0
Deceleration *	0 or 1 *	SignalSense *	motor dependent *
DerivativeTime *	0 or 1 *	Start Velocity *	0 *
EncoderModulus	0	Step Range	1
Encoder Source	motor dependent *	StopMode	0
EncoderToStepRatio	00010001h	TraceMode	0
GearMaster	0	TracePeriod	1
GearRatio	0	TraceStart	0

Holding Motor Limit *	32767 *	TraceStop	0
Holding Delay *	motor dependent *	TraceVariable1	0
IntegrationLimit	0	TraceVariable2	0
InterruptMask	0	TraceVariable3	0
Jerk	0	TraceVariable4	0
Kaff	0	TrackingWindow	0
Kd	0	Update Mask *	0Bh *
Ki *	65535 or 0 *	Velocity	0
Kout	65535		

Die Motorabhängigen Standardwerte sind wie folgt:

Variable	DC Brushed	DC Brushless (3 phase)	DC Brushless (2 phase)
Actual Position Units *	0	0	0
CommutationMode	n/a	0	0
EncoderSource	0	0	0
Motion Error Event Action *	5	5	5
OutputMode	1	2	2
PhaseCorrectionMode	n/a	1	1
PWMFrequency (kHz)	0	n/a	n/a
Phase Counts *	n/a	1	1
Holding Delay *	n/a	n/a	n/a
Signal Sense *	0	0	0

Variable	Microstepping (3 phase)	Microstepping (2 phase)	Pulse & Direction
Actual Position Units *	1	1	1
CommutationMode	0	0	n/a
EncoderSource	2	2	3
Motion Error Event Action *	0	0	0
OutputMode	2	1	n/a
PhaseCorrectionMode	n/a	n/a	n/a
PWMFrequency (kHz)	20	80	n/a
Phase Counts *	256	256	n/a
Holding Delay *	32767	32767	20
Signal Sense *	0	0	0800h

Anmerkungen:

*BiQuadCoefficient: nicht verfügbar für POSYS® 7xx, 8xx, 8xx-B und 9xx

BufferStart:
Der Standardwert nach einem Reset für die POSYS® Serien 7xx, 8xx, 8xx-B and 9xx ist 200h. I kleinerer Wert ist nicht möglich.

Der Standardwert nach einem Reset für die POSYS® Serien 18xx und 19xx ist 0h.

PositionErrorLimit:
Der Standardwert nach einem Reset für die POSYS® Serien 7xx, 8xx, 8xx-B and 9xx ist 2147483647.
Der Standardwert nach einem Reset für die POSYS® Serien 19xx and 18xx ist 65535.

SampleTime:
Hier verweisen wir auf die Beschreibung zum Kommando Set/GetSampleTime.

Alle unterstützten Achsen der Karte sind wieder aktiviert nach dem Reset.

Profile, Servofilter und andere achsen-spezifischen Parameter werden resettet.

On-Board Speicherbufferparameter werden resettet. BufferStart ist zurückgesetzt auf Adresse 200h für die POSYS® 7xx/8xx/8xx-B und 9xx. Für die POSYS® 18xx/19xx ist die BufferStartAdresse auf 0h zurückgesetzt.

Achsen-spezifische Bedingungen werden auf allen Achsen resettet. On-board-SpeicherBuffer Bedingungen sind für alle 15/32 Speicherbuffer zurückgesetzt.

Mit "*" versehene Parameter sind in der Regel nur beim POSYS® 1800/1900 verfügbar, dabei g

	es auch noch Unterschiede zwischen den Versionen bis 1.5 und ab 2.x. Der 2. Wert steht für Versionen ab 2.x. Steht nur ein Wert gibt es diese Resetoption nur in Versionen ab 2.x (Ausnah BiquadCoefficients; in allen Magellanversionen verfügbar; ausser Versionen mit Piezo-Funktionalität)
Einschränkung	PWMFrequency kann nicht für die POSYS® 7xx, 8xx, 8xx-B und 9xx gesetzt werden.
	Warnung: Falls der externe Reset für einen längeren Zeitraum permanent auf "low" gezogen wird, kann der Motion Prozessor beschädigt werden. Es ist absolut notwend und es liegt in der Verantwortung des Anwenders eine Schaltung für den externen R zu implementieren, welche nicht einen permanenten Reset zur Folge hat, auch dann nicht wenn ein kompletter Maschinennotstop v orliegt. In diesem F all erlischt jedwe Garantie oder Gewährleistungsanspruch.
siehe	
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)
MotionScript® Aufruf-konvention	HardReset(hPosys#) oder MCReset(hPosys#)

¤

NoOperation

Syntax	NoOperation(hPosys) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
Definition	Das NoOperation Kommando hat keinen anderen Effekt auf die POSYS® als die Kommunikation zu prüfen.			
Einschränkung				
siehe				
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	NoOperation(hPosys#)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

PortInW
PortOutW

Syntax	PortInW(hPosys, address) stdcall PortOutW(hPosys, address, data) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
		Type		
	address	unsigned 16 bits	Range	51-1023
			Scaling	none
	data	unsigned 16 bits	Range	0-65535
			Scaling	none
Definition	Unter normalen Bedingungen sollten die Kommandos PortInW und PortOutW nicht verwendet werden. Falls eine falsche Verwendung stattfindet, kann es zu einem C rash bis Zerstörung des Computer-Systems führen, inkl. Runaway der Motoren. Diese Kommandos sollten nur von erfahrenen Programmierern verwendet werden. Diese Funktionen unterstützen direkten Zugriff die Adressbereiche die vom Treiber geöffnet wurden. PortInW liest ein Datenwort von der Adres PortOutW schreibt ein Datenwort an die Adresse. Diese Kommandos erlauben erfahrenen Programmierern sehr hardwarenahe Programmierung.			
Einschränkung	Benutzen Sie diese Kommandos nicht, außer wenn Sie vollkommen sicher sind zu wis was Sie tun!!!			
siehe	OpenDevice/CloseDevice			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	PortInW(hPosys#, address) PortOutW(hPosys#, address, data)			

¤

ReadAnalog

Syntax	ReadAnalog(hPosys, portID) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	Name	Type		
	portID	unsigned 16 bits	Range	0 to 7
			Scaling	unity
			Units	-
Ergebnis	Name	Type		
	value	unsigned 16 bits	Range	0 to 2^{16}-1
			Scaling	$100/2^{16}$
			Units	% input
Definition	ReadAnalog liefert einen 16-Bit Wert der die Spannung (gelesen von einem 10-Bit A/D Wandler) für den spezifizierten analogen Eingang entspricht. Der Wert der zurückgeliefert wird, ist das Ergebnis einer Verschiebung des 10-Bit Wertes nach links um 6 Bit.			
Einschränkung				
siehe				
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	ReadAnalog(hPosys#, portID)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

ReadIO

Syntax	ReadIO(hPosys, address) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	Name	Type		
	address	unsigned 16 bits	Range	0 to 255
			Scaling	unity
			Units	-
Ergebnis	Name	Type		
	data	unsigned 16 bits	Range	0 to 2^{16}-1
			Scaling	unity
			Units	-
Definition	ReadIO liest ein 16-Bit Datenwort vom Gerät dessen Adresse kalkuliert wird indem zur Adresse 1000H addiert wird. (address ist ein Offset der base address, 1000h, der POSYS® memory-mapped I/O space). Das Format und die Interpretation des 16-Bit Datenwortes sind abhängig vom benutzerdefiniert Gerät dass adressiert wird. Benutzerdefinierte I/O können verwendet werden eine Vielzahl zusätzliche Funktionen hinzu zu fügen, wie z.B. Parallel I/O, Flash Memory für nichtflüchtige Konfigurationsinformationen oder Ausgabegeräte wie z.B. LED Arrays.			
Einschränkung				
siehe	WriteIO			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	ReadIO(hPosys#, address)			

¤

ReadPLD

Syntax	ReadPLD(hPosys) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
Ergebnis	Name	Type		
	value	unsigned 16 bits	Range	0 to 2^{15}-1
			Scaling	unity
			Units	-
Definition	Liest interne Systemdaten bezgl. Konfiguration und Revisionsstufe.			
Einschränkung				
siehe				
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	ReadPLD(hPosys#)			

¤

Set/GetCANMode

SetCANMode
GetCANMode

nur verfügbar für POSYS® 18xx/19xx Series

Syntax	SetCANMode(hPosys, mask) stdcall GetCANMode(hPosys) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
		Type		
	mask	unsigned 16 bit	siehe unten	

Definition

SetCANModesetzt die CAN 2.0B Kommunikationsparameter für die POSYS®. Nach der Ausführung dieses Kommandos wird die POSYS® auf eine an der Adresse 0x600 + nodeID empfangene Nachricht antworten. C AN-Antworten werden an die Adresse 0x580 + nodeID geschickt. Die Übertragungsrate wird mit dem Parameter für die Übertragungsrate definiert.

Die folgende Tabelle listet die möglichen Parameter für dieses Kommando auf:

Bit Number	Name	Instance	Encoding
0-6	CAN node ID	Address 0 Address 1 ... Address 127	0 1 ... 127
7-12	reserved		
13-15	transmission rate	1,000,000 baud 800,000 500,000 250,000 125,000 50,000 25,000 10,000	0 1 2 3 4 5 6 7

Einschränkung	
siehe	
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)
MotionScript® Aufruf-konvention	SetCANMode(hPosys#, mask) GetCANMode(hPosys#)

¤

Set/GetDiagnosticPortMode

SetDiagnosticPortMode
GetDiagnosticPortMode

nur verfügbar für POSYS® 8xx-B/9xx Series

Syntax	SetDiagnosticPortMode(hPosys, mask) stdcall GetDiagnosticPortMode(hPosys) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
		Type		
	mode	limited full	0 1	
Definition	SetDiagnosticPortMode bestimmt welche Kommandos über den Diagnosticport ausgeführt werden können. Wenn gesetzt auf "limited" können nur die folgenden Kommandos ausgeführt werden: alle GET Kommandos Das SetBufferReadIndex Kommando Wenn eingestellt auf “full” können alle Kommandos ausgeführt werden. GetDiagnosticPortMode liest den programmierten Wert zurück.			
Einschränkung	nur gültig für POSYS® 8xx-B and 9xx series			
siehe	Set/GetSerialPortMode			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetDiagnosticPortMode(hPosys#, mask) GetDiagnosticPortMode(hPosys#)			

¤

Set/GetSerialPortMode

SetSerialPortMode
GetSerialPortMode

nur verfügbar für POSYS® 8xx-B/9xx/18xx/19xx Series

Syntax

SetSerialPortMode(hPosys, mask) stdcall
GetSerialPortMode(hPosys) stdcall

Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction

Argumente

Name	Instance	Encoding
hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice

Name	Type	Encoding
mask	unsigned 16 bits	siehe unten

Ergebnis

Definition

SetSerialPortMode konfiguriert die asynchrone, serielle Schnittstelle.

Anmerkung: Es wird empfohlen 2 Stoppbits zu verwenden, falls Übertragungsgeschwindigkeiten höher als 19200 Baud verwendet werden.

GetSerialPortMode liest die programmierte Konfiguration zurück.

Bit number	Name	Instance	Encoding
0-3	Transmission rate	1200 baud	0
		2400	1
		9600	2
		19200	3
		57600	4
		115200	5
		250000	6
		416667	7
4-5	parity	none	0
		odd	1
		even	2
6	stop bits	1	0
		2	1
7-8	protocol	point-to-point	0
		multi-drop using address bit	1
		multi-drop using idle line detection	2
11-15	Multi-drop address	Address 0	0
		Address 1	1
		...	...
		Address 31	31

Einschränkung

Für die POSYS® 8xx-B und 9xx muss das Kommando Set/GetDiagnosticPortMode ausgeführt werden um vollen Zugriff auf alle Kommandos zu erhalten wenn Zugriff über die serielle Schnittstelle erfolgen soll. Zusätzlich muss noch eine Steckbrücke gesetzt werden. Sehen Sie hierzu im Kapitel "Benutzerhandbuch/POSYS Versionsspezifikationen/POSYS® 8xx-B oder POSY 9xx für weitere Informationen nach.

Nicht verfügbar für POSYS® 7xx, 8xx.

Set/GetDiagnosticPortMode ist nicht verfügbar für die POSYS® 1800/1900 Serien. Diese Karten verfügen generell über die Möglichkeit voller Kommunikationsfähigkeit über alle 3 möglichen Schnittstellen (PC-Bus, seriell, CAN)

siehe

Set/GetDiagnosticPortMode

DLL: Delphi, C++, VB

wir verweisen auf die verschiedenen Importdateien (imp*.*)

MotionScript® Aufruf-konvention

SetSerialPortMode(hPosys#, mask)
GetSerialPortMode(hPosys#)

¤

Set/GetSynchronizationMode

SetSynchronizationMode
GetSynchronizationMode

nur verfügbar für POSYS® 8xx-B/9xx mit SYNC IO Option und 18xx/19xx Serie als Standard

Syntax	SetSynchronizationMode(hPosys, mode) stdcall GetSynchronizationMode(hPosys) stdcall		
Motortyp	DC Brush	DC Brushless	Microstepping
Argumente	Name	Instance	Encoding
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice
	mode	Disabled Master Slave	0 1 2
Definition	SetSynchronizationMode setzt den Modus des SYNC-IOs, das verwendet wird für die Synchronisation des internen Zeitgebers über mehrere POSYS® Motion Controller hinweg. Im deaktivierten Modus, ist der IO als Eingang konfiguriert und wird nicht verwendet. Im Master-Mode gibt der Pin einen Synchronisationstakt aus, dass von mehreren Slave-Karten verwendet wird den internen Arbeitstaktzyklus mit dem Master zu synchronisieren. Im Slave-Modus wird der Pin als Eingang verwendet und ein Takt auf dem Eingang synchronisiert den internen Arbeitszyklus mit dem Master. Die Synchronisation kann so auf 10 µsec. genau erfolgen. GetSynchronizationMode liest den programmierten Modus zurück.		
Einschränkung	Falls irgend eine Achse für den Schrittmotorbetrieb konfiguriert wurde, kann dieses Kommando nicht mehr verwendet werden. Der SYNC IO ist nicht verfügbar für folgende POSYS®: 7xx, 8xx, 85x-B, 95x. Für die POSYS® 18xx oder 19xx ist der SYNC I/O deaktiviert falls eine Achse im Schrittmotormodus betrieben wird.		
siehe	Set/GetSampleTime, Set/GetBreakpoint, Set/GetBreakpointValue		
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)		
MotionScript® Aufruf-konvention	SetSynchronizationMode(hPosys#, mode) GetSynchronizationMode(hPosys#)		

¤

SetWatchDog

Syntax	SetWatchDog(hPosys,address,value) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	address		0x04	
	value		0x5562	
Definition	SetWatchDog aktiviert einen Timer auf der POSYS® welches wenn 104 Millisekunden überschritten werden, einen HardReset der Karte ausführt, ausser der Timer wurde vorher angesprochen. Der Watchdog Timer ist nach einem Timeout deaktiviert und muss neu initialisiert werden. Beispiel: address = 4 Rem 0x04 timeout = 21858 Rem 0x5562 SetWatchDog(hPosys,address,timeout) Dies aktiviert den Watchdog Timer. Es muss alle 104 Millisekunden angesprochen werden, ansonsten wird ein HardReset ausgeführt! Adress und Timeout Werte können und dürfen nicht verändert werden!!!			
Einschränkung	Der Watchdog Timer kann nachdem einmal aktiviert ohne ein Reset der Karte nicht deaktiviert werden. Der Timeout ist fix auf 104 Millisekunden eingestellt.			
siehe	OpenDevice/CloseDevice, PortInOutW			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	SetWatchDog(hPosys#,address,value)			

¤

WriteIO

Syntax	WriteIO(hPosys, address, data) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	POSYS® 18xx, 19xx	Type	Range	
	address	unsigned 16 bits	0 to 255	
	data	unsigned 16 bits	0 to 2^{16}-1	
	POSYS® 7xx, 8xx, 8xx-B, 9xx	Type	Range	
	address	unsigned 8 bits	0 to 255	
	data	unsigned 16 bits	0 to 2^{16}-1	
Ergebnis				
Definition	WriteIO schreibt ein 16-Bit Datenwort in ein Gerät dessen Adresse sich daraus errechnet indem 1000h der Adresse hin zu addiert wird. (address ist ein Offest der Basisadresse, 1000h, vom memory-mapped I/O Bereich der POSYS®). Das Format und die Interpretation des 16-Bit Datenwortes sind abhängig vom benutzerdefiniert Gerät dass adressiert wird. Benutzerdefinierte I/O können verwendet werden, eine Vielzahl zusätzliche Funktionen hinzuzufügen, wie z.B. Parallel I/O, Flash Memory für nichtflüchtige Konfigurationsinformationen oder Ausgabegeräte wie z.B. LED Arrays.			
Einschränkung				
siehe	ReadIO			
DLL: Delphi, C++, VB	wir verweisen auf die verschiedenen Importdateien (imp*.*)			
MotionScript® Aufruf-konvention	WriteIO(hPosys#, address, data)			

¤

Zusätzliche MotionScript® Funktionen

BitReset

BitReset

Syntax	BitReset(hPosys#, testVar, bitToReset) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys#	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	testVar	32 bits		
	bitToReset	32 bits		
Definition	BitReset resettet bitToReset (zu Null gesetzt) in testVar. Result = BitReset(testVar, bitToReset)			
Einschränkung	Nur verfügbar in der MotionScript® IDE			
siehe	BitTest, BitSet, CompareActivity, CompareEvent, CompareSignal			
DLL: Delphi, C++, VB	nicht verfügbar			
MotionScript® Aufruf-konvention	BitReset(hPosys#, testVar, bitToTest)			

¤

BitSet

Syntax	BitSet(hPosys#, testVar, bitToSet) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys#	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	testVar	32 bits		
	bitToTest	32 bits		
Definition	BitSet setzt bitToSet in testVar indem 2 Werte mit OR verglichen werden. Result = BitTest(testVar, bitToSet)			
Einschränkung	Nur verfügbar in der MotionScript® IDE			
siehe	BitReset, BitTest, CompareActivity, CompareEvent, CompareSignal			
DLL: Delphi, C++, VB	nicht verfügbar			
MotionScript® Aufruf-konvention	BitSet(hPosys#, testVar, bitToTest)			

¤

© POSYS Motion Control GmbH & Co.KG, 2013

BitTest

Syntax	BitTest(hPosys#, testVar, bitToTest) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys#	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	testVar	32 bits		
	bitToTest	32 bits		
Definition	BitTest liefert eine 1 falls der bitToTest gesetzt ist, ansonsten liefert es eine Null. Result = BitTest(testVar, bitToTest)			
Einschränkung	Nur verfügbar in der MotionScript® IDE			
siehe	BitReset, BitSet, CompareActivity, CompareEvent, CompareSignal			
DLL: Delphi, C++, VB	nicht verfügbar			
MotionScript® Aufruf-konvention	BitTest(hPosys#, testVar, bitToTest)			

¤

CompareActivity

Syntax	CompareActivity(hPosys#, axis, mask) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys#	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	mask	0 - 65535 (0xFFFF)		
Definition	CompareActivity liefert das Ergebnis einer AND Operation der BitMask mitGetActivityStatus. Result = (GetActivityStatus und mask)			
Einschränkung	Nur verfügbar in der MotionScript® IDE			
siehe	BitReset, BitTest, BitSet, CompareEvent, CompareSignal			
DLL: Delphi, C++, VB	nicht verfügbar			
MotionScript® Aufruf-konvention	CompareActivity(hPosys#, axis, mask)			

¤

CompareEvent

Syntax	CompareEvent(hPosys#, axis, mask) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys#	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	mask	0 - 65535 (0xFFFF)		
Definition	CompareActivity liefert das Ergebnis einer AND Operation der BitMask mitGetEventStatus. Result = (GetEventStatus und mask)			
Einschränkung	Nur verfügbar in der MotionScript® IDE			
siehe	BitReset, BitTest, BitSet, CompareActivity, CompareSignal			
DLL: Delphi, C++, VB	nicht verfügbar			
MotionScript® Aufruf-konvention	CompareEvent(hPosys#, axis, mask)			

¤

CompareSignal

Syntax	CompareSignal(hPosys#, axis, mask) stdcall			
Motortyp	DC Brush	DC Brushless	Microstepping	Pulse & Direction
Argumente	Name	Instance	Encoding	
	hPosys#	CardHandle	Wert zurückgeliefert durch Aufruf von OpenDevice	
	axis	Axis1 Axis2 Axis3 Axis4	0 1 2 3	
	mask	0 - 1024 (0x0400)		
Definition	CompareActivity liefert das Ergebnis einer AND Operation der BitMask mitGetSignalStatus. Result = (GetSignalStatus und mask)			
Einschränkung	Nur verfügbar in der MotionScript® IDE			
siehe	BitReset, BitTest, BitSet, CompareActivity, CompareEvent			
DLL: Delphi, C++, VB	nicht verfügbar			
MotionScript® Aufruf-konvention	CompareSignal(hPosys#, axis, mask)			

¤

Kommandoübersicht - ReadIO/WriteIO

ReadIO / WriteIO

ReadIO, WriteIO

Zugriff auf Karten-spezifische Funktionalitäten erhält man durch das für allgemeine Zwecke gedachte I/O-Adressierungsschema. Die Kommandos ReadIO und WriteIO erlauben den Zugriff auf diese spezifischen Funktionen, die von der POSYS® unterstützt werden. Für weitere Informationen bzgl. der ReadIO/WriteIO Kommandos verweisen wir auf die Zusatzbeschreibung in der Sektion FAQs/Programmer's Info und auf die Beschreibung der beiden Kommandos im Programmierhandbuch (ReadIO/WriteIO).

Die folgende Tabelle listet die verschiedenen Adressierungen und dadurch erhältlichen Funktionen auf:

I/O Space Address Offset	Register	Beschreibung
+0	digital I/O	Dieses Register erlaubt das Schreiben und Lesen dieser 8 für allgemeine Zwecke gedachten digitalen Ein- und Ausgänge.
+1	Amplifier & DAC Enable	Dieses Register erlaubt das Schreiben und verifizieren der Ausgangssignale für die Verstärkerfreigaben 1-4. Diese Ausgänge können auch für allgemeine Zwecke verwendet werden. Ebenso werd mit diesem Register die DAC Enable Ausgänge beschrieben und verifiziert.
+2	Reset Monitor	Register um die Überwachung zu checken und zu resetten
Not used	nicht benutzt	
+4	Watchdog	Register um die Watchdog Funktion zu aktivieren und upzudaten.
+0xff	Card ID	Register um das KartenID-Wort zu lesen.

Digital IO Kontrollregister (I/O space address +0)

Die folgende Tabelle beschreibt Details für das +0 Digital IO Kontrollregister.

I/O Adresse	Bitwert	Signale
+0	0-7	Digitale Ausgangssignale 0-7
	8-15	Dgitale Eingangssignale 8-15

Verstärker- & DAC Freigabe Kontrollregister (I/O space address +1)

Die folgende Tabelle beschreibt die Details für das +1 Verstärker- & DAC Freigabe Kontrollregister.

I/O Adresse	Bitwert	Signale
+1	0-3	Verstärkerfreigabeausgänge (0-3)
	4-6	nicht benutzt
	7	DAC Freigabestatus (1 = aktiviert, 0 = deaktiviert)
	8-11	Maske ändern für Bits 0-3 - Verstärkerfreigabeausgänge (1 = änderr 0 = nicht ändern)
	12-14	nicht benutzt
	15	Maske ändern für DAC Freigabe (1 = ändern, 0 = nicht ändern)

Reset Überwachungskontrollregister (I/O space address +2)

Die folgende Tabelle beschreibt die Details für das +2 address Reset Überwachungskontrollregister.

I/O Adresse	Bitwert	Signale
+2	0-11	reserviert
	12	Softwarekommando: eine 1 in diesem Bit zeigt an, dass ein Reset durch ein Anwendersoftwarekommando ausgelöst wurde.

I/O Adresse	Bitwert	Signale
	13	Under voltage detection: eine 1 in diesem Bit zeigt an, dass ein Rese aufgrund einer Unterspannungserkennung ausgelöst wurde.
	14	External signal: eine 1 in diesem Bit zeigt an, dass ein Reset aufgrun eines externen Signals ausgeführt wurde. Ein solches Reset kann z.B über das GP C on herrühren, dass mit dem externen Resetknopf des IO700/800 verbunden ist. Wenn dieses Signal auf „low“ geht, wird ei Reset ausgelöst.
	15	Watchdog timeout: eine 1 in diesem Bit zeigt an, dass ein Reset aufgrund eines Watchdog Timeouts erfolgte.

Card ID Kontrollregister (I/O space address +0xff)

Die folgende Tabelle beschreibt die Details für das +0xff address card ID Register

I/O Adresse	Bitwert	Signale
+0xff	0-3	Major PLD Revision: binär-kodiertes 4-Bit Wort, welches die Major PL Revision angibt. Dieser Wert reicht von 0-15.
	4-7	Minor PLD Revision: binär-kodiertes 4-Bit Wort, welches die Minor PL Revision angibt. Dieser Wert reicht von 0-15.
	8-11	Board revision: binär-kodiertes 4-Bit Wort, welches die Board Revisi angibt. Dieser Wert reicht von 0-15.
	12-15	Boardtyp: binär-kodiertes 4 Bit Wort, welches die Boardtype angibt. Dieser Wert kann eine der folgende Werte haben: 0 – ISA-Bus basierte Kartenfamilie 1 – PCI-Bus basierte Kartenfamilie 2 – unbenutzt 3 – PC/104 basierte Kartenfamilie 4 – 15 unbenutzt

Watchdog Kontrollregister (IO space address +4)

Die folgende Tabelle beschreibt die Details für das +4 address Watchdog Timeout Register. Ein Scheibkommando mit WriteIO aktiviert den Watchdog Timer. Anschliessend muss alle 104 millisec. in dieses Register geschrieben werden um ein Reset des Controllers zu vermeiden.

I/O Adresse	Bitwert	Daten
+4	0-15	0x5562

¤

Application Notes

Extension Port Connector J6 (POSYS® 700, ISA-Bus)

Extension Port Connector J6 (POSYS® 700, ISA-Bus)

Pin No.	Description	Pin No.	Description
1	GND	2	A0
3	DS0	4	A1
5	DS1	6	A2
7	DS2	8	A3
9	DS3	10	A4
11	DS4	12	A5
13	DS5	14	A6
15	DS6	16	A7
17	DS7	18	A8
19	DS8	20	A9
21	DS9	22	A10
23	DS10	24	A11
25	DS11	26	A12
27	DS12	28	A13
29	DS13	30	A14
31	DS14	32	A15
33	DS15	34	IS-
35	DS-	36	STRB-
37	WE-	38	VCC
39	R/W	40	PS-

DS0..DS15: data

A0..A15 : addresses

IS-, DS-, STRB-, WE-, R/W, PS- : connections to CP

Connector J9

The analog inputs to this connector are in parallel to the analog inputs of connector J7.
J9 provides also inputs for external voltage references (high and low) selected by jumpers JP13, JP14 and JP15.
For example Analog1 from J7 is tied to Analog1 of J9 and Analog8 of J7 is tied to Analog8 of J9.

Date: 09/09/03
Servo Halbeck GmbH & Co. KG

¤

Stall Detection

Subject: POSYS® Series, Stepping Motor Stall Detection

Stepping motor versions of the POSYS® series controllers can be equipped with encoder feedback. This provides means for monitoring the actual position of the axis. It also provides means of detecting a stall condition of the stepping motor. In order to activate this feature following command must be issued to the POSYS®:

- SetEncoderSource(axis, incremental) //incremental = 0
- SetEncoderToStepRatio(axis, counts, steps) //for example 400 encoder counts, 100 motor steps
- SetPositionErrorLimit(axis, limit)
- ClearPosition Error(axis) //clears any follow error
- Update(axis)
- SetAutoStopMode(axis, On) //On = 1, Off = 0; this stops the motor if the limit is exceeded

The above settings would stop the motor if the difference between commanded and actual position would exceed 500 steps. The command SetEncoderToStepRatio will not change the value returned by GetActualPosition. In the above example. A move of 500 motor steps would equal 2000 encoder counts. Assuming no counts would be generated but the motor would turn, then it would require 2000 encoder counts (this corresponds to 500 x 4 steps) to deactivate the motion generator, at this point no steps would be output if a start command would be issued.

If in the above example GetActualPosition(axis) returns 228, the command

ClearPositionError(axis) would copy the value 57 (228 / 4) to the Commanded Position register.

Date: 26.01.2004
Revised:

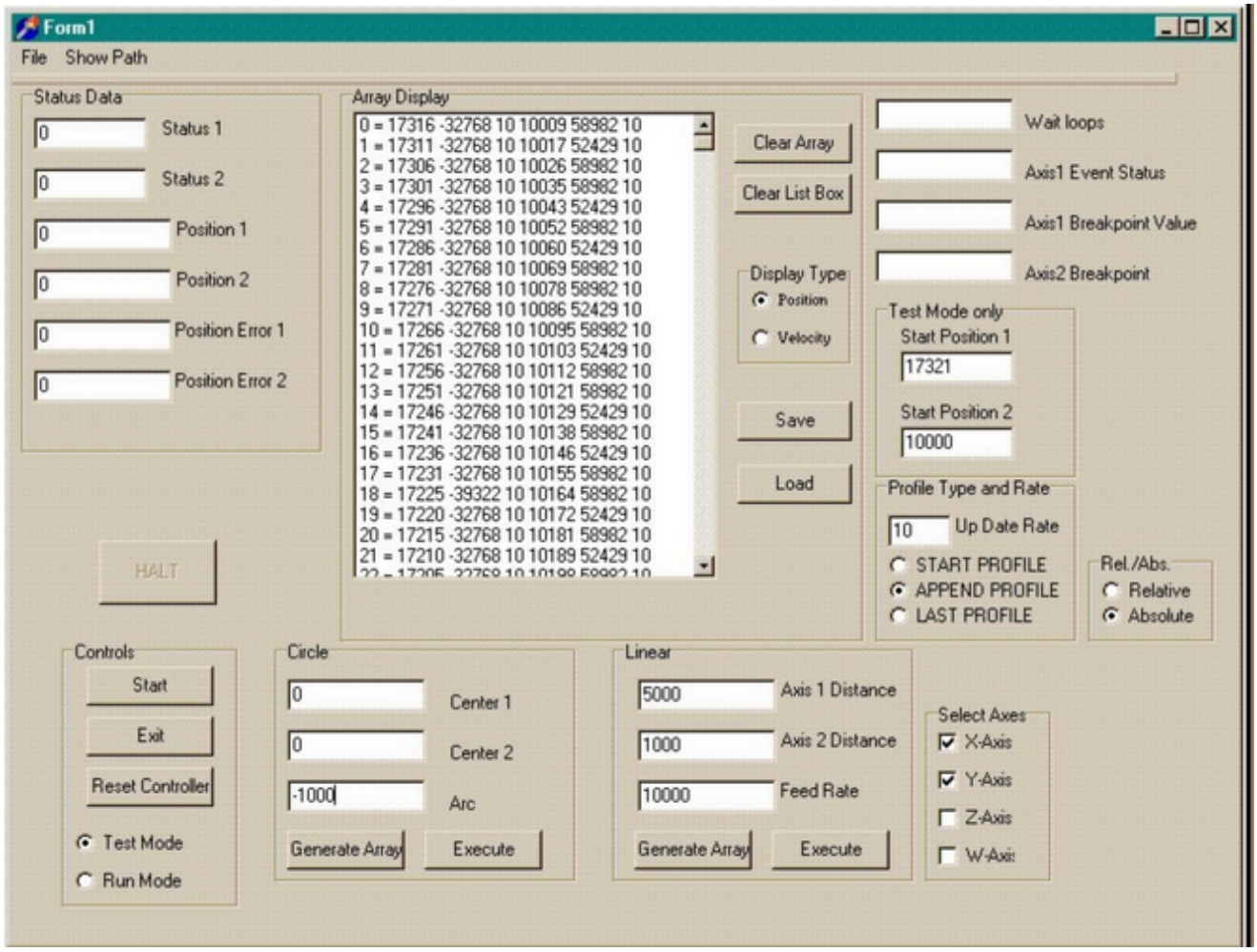
¤

Coordinated Motion with POSYS® Contouring Library

Theme: Generating coordinated motion data

The POSYS® series supports on-board RAM. This RAM can be used for trace buffer and/or program buffer. If complex motion is required then we recommend the use of the on-board RAM. Data points, position, velocity and an Update-multiplicator need to be generated and stored in the on-board RAM. The program CONTOUR is a help for simple 2-axes coordinated motion. This document describes the usage of this program.

- Install the POSYS® main program with the integrated Setup for Windows 9x/ME/NT/2000/XP.
- After installation change to the new created subdirectory /Driver Installation. Start the program Ksetup.exe.
- Change to the new created subdirectory /lib and start the batch-file registerOCX.exe.
- Decompress this program to a subdirectory of your choice and start the program.

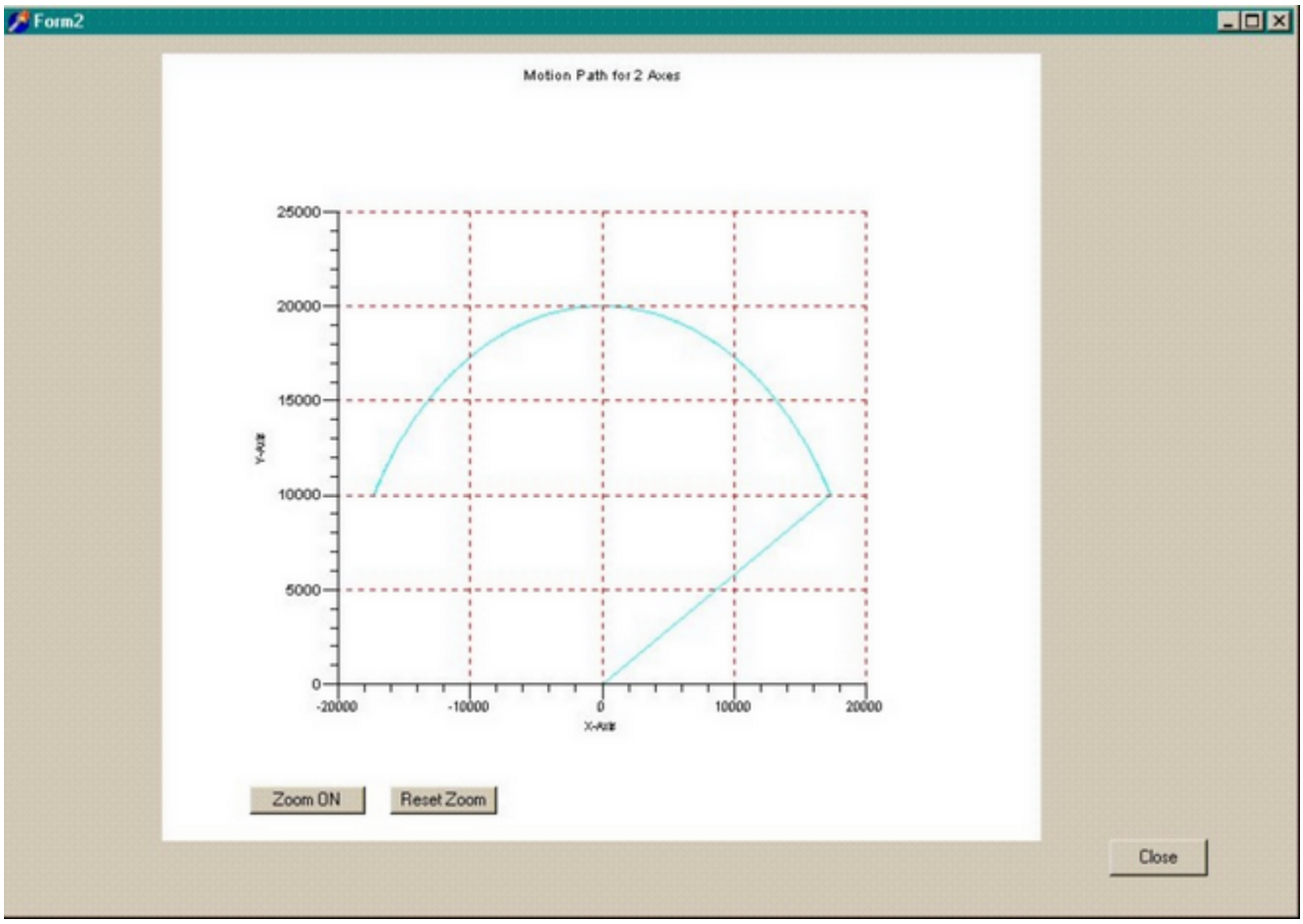


Screen with Contour Data

Program description

1. Controls Box: choose Test Mode. Push Start.
2. The box “TestMode only” opens.
3. Input in the windows “StartPosition1” and “StartPosition2” the start positions of the axes 1 and 2. Defaults are 0. The upper example shows values for the X-axis = 17321 and for the Y-axis = 10000.
4. In the Box “SelectAxes” the X and Y axes are set to default. You can choose any other possible configuration.
5. In the Box “Profile Type and Rate” choose START PROFILE and the desired Up Date Rate. Up Date Rate is a factor with which the Chip Update Time is multiplied and then valid is for each segment. Assumed you have a 4-axes card (servo or stepper), then the Chip Update Time = 0.0004096 seconds. This value is multiplied with the Chip Update Rate, in this example with 10. Each segment will be driven with the calculated speed of increments/Chip Update Time. Each segment in the example has therefore a time frame of 4.096 milliseconds.

6. The Box "Linear" will be used for coordinated linear motion and for the desired velocity (Feed Rate) in Millicounts/(Chip Update Time). The value 10000 Millicounts = 10 counts/(Chip Update Time) = 24 414 pulses/second. Axis 1 Distance and Axis 2 Distance are occupied by defaults, which need to be changed if necessary. For our example we took the value 10000 for FeedRate.
7. The Box Circle serves for the input of data for circular interpolation. Center 1 and Center 2 contain the values for the center of the circle and Arc the value for the circular arc in millidegrees. A CW motion is positive and CCW motion is negative. For a CW circular arc of 15,54 degrees you must input 15540.
8. For the first segment which will be calculated START PROFILE must be chosen first. For all additional segments APPEND PROFILE must be defined. The last segment can be either defined as APPEND PROFILE or LAST PROFILE.
9. Then the button Generate Array for either linear or circular motion calculation must be pushed to calculate the path information. The generated data will be shown in the windows Array Display.
10. To save the data you can click the button Save. To load the data you can click the button Load.
11. The menu point Show Path/Display allows to graphically display a calculated path. The screen shot below shows a typical motion. The center of the circle is at (0,0), the motion will be -120 degrees, the radius is 20000. The starting point for X=17321 and Y=10000.



Circular Motion

12. With Zoom ON and using the right button of the mouse you can choose an area of the path which should be shown in a different zoom factor. Reset Zoom or doubleclick of the right mouse button resets the display to the default size.
13. In the Box Rel/Abs. You can choose if the input data is absolute or relative. At the moment only absolute can be chosen.
14. The button Clear Array clears a programmed array, the button Clear List Box clears the content of the output window.

Sample programming

The values in the above screen shot shall be used for this programming example. All necessary steps will be shown.

1. The two axes are at position: X = 17321 (Start Position 1), Y = 10000 (Start Position 2) which are considered the start positions. Input in Test Mode Box.
2. Velocity (Feed Rate) = 10000 Millicounts. Input in Linear Box.
3. Motion path = -120000 millidegrees (120 degrees CCW), radius 20000, center of circle (0,0), Input as Center 1 and Center 2.
4. Linear acceleration and deceleration over 9 degrees each with start at 1000 Millicounts. The acceleration is not implemented in the this version yet and needs to be done manually by deviding the segment into 1 degree motions.
5. Input:

START PROFILE
Start Position 1 = 17321
Start Position 2 = 10000
Feed Rate = 1000
Center 1 = 0
Center 2 = 0
Arc = -1000 (equals 1 degree)
button Generate Array in Box Circle
APPEND PROFILE
Feed Rate = 2000
Center 1 = 0
Center 2 = 0
Arc = -1000 (equals 1 degree)
button Generate Array in Box Circle
Feed Rate = 3000
Center 1 = 0
Center 2 = 0
Arc = -1000 (equals 1 degree)
button Generate Array in Box Circle
Feed Rate = 4000
Center 1 = 0
Center 2 = 0
Arc = -1000 (equals 1 degree)
button Generate Array in Box Circle
Feed Rate = 5000
Center 1 = 0
Center 2 = 0
Arc = -1000 (equals 1 degree)
button Generate Array in Box Circle
Feed Rate = 6000
Center 1 = 0
Center 2 = 0
Arc = -1000 (equals 1 degree)
button Generate Array in Box Circle
Feed Rate = 7000
Center 1 = 0
Center 2 = 0
Arc = -1000 (equals 1 degree)
button Generate Array in Box Circle
Feed Rate = 8000
Center 1 = 0
Center 2 = 0
Arc = -1000 (equals 1 degree)
button Generate Array in Box Circle
Feed Rate = 9000
Center 1 = 0
Center 2 = 0
Arc = -1000 (equals 1 degree)
button Generate Array in Box Circle
Feed Rate = 10000
Center 1 = 0
Center 2 = 0
Arc = -102000 (equals 102 degrees with constant velocity)
button Generate Array in Box Circle
Feed Rate = 9000
Center 1 = 0
Center 2 = 0
Arc = -1000 (equals 1 degree)
button Generate Array in Box Circle
Feed Rate = 8000
Center 1 = 0
Center 2 = 0
Arc = -1000 (equals 1 degree)
button Generate Array in Box Circle
Feed Rate = 7000
Center 1 = 0
Center 2 = 0
Arc = -1000 (equals 1 degree)
button Generate Array in Box Circle
Feed Rate = 6000
Center 1 = 0
Center 2 = 0
Arc = -1000 (equals 1 degree)
button Generate Array in Box Circle

Feed Rate = 5000
Center 1 = 0
Center 2 = 0
Arc = -1000 (equals 1 degree)
button Generate Array in Box Circle
Feed Rate = 4000
Center 1 = 0
Center 2 = 0
Arc = -1000 (equals 1 degree)
button Generate Array in Box Circle
Feed Rate = 3000
Center 1 = 0
Center 2 = 0
Arc = -1000 (equals 1 degree)
button Generate Array in Box Circle
Feed Rate = 2000
Center 1 = 0
Center 2 = 0
Arc = -1000 (equals 1 degree)
button Generate Array in Box Circle
Feed Rate = 1000
Center 1 = 0
Center 2 = 0
Arc = -1000 (equals 1 degree)
button Generate Array in Box Circle

The input and generating of the Array is complete. Now the data needs to be uploaded into the on-board RAM and executed. The motion will be a arc of 120 degrees.

Datum: 15.09.2003

Revision:

Europe: support@halbeck.com

POSYS® is a registered trademark of POSYS Motion Control

¤

Coordinated Moves using POSYS® 900 Motion Controller

There are different ways of programming a coordinated move. This description refers to standard POSYS900.DLL functions.

- Calculate speed and acceleration values for every axis and do a MultiUpdate(axes) (simultaneous start).
- Assign Master and Slave axis and program the Ratio
- Use the on-board RAM to store a motion profile and execute this sequence of commands
- For simple linear moves vector speed and vector acceleration must be known as well as the destination points for the involved axes. The routines called from the DLL will then calculate the corresponding axis speeds and acceleration.
  a. LinearMove2(axis1, axis2, vectorspeed, vectoracceleration, distance1, distance2)
  b. Axis1 = 0 3
  c. Axis2 = 0 3
  d. vectorspeed = quadrature counts/sec
  e. vectoracceleration = quadrature counts/sec²
  f. distance1, distance2 = quadrature counts
- Similar calls exists for 3 and 4 axes linear moves like LinearMove3( ) and LinearMove4( )
  a. It is possible to preload new speeds, acceleration and distances on the fly and activate (update) these values automatically using the SetBreakpoint and SetBreakpointValue functions, an automatic update will occur. This can now be set to generate an interrupt to alert the host that an interrupt occurred and new data may be loaded for the next update.
  b. Depending on the speed of the host computer update rates of 2 milliseconds or faster are possible. If the fast MC (version 2.0 and higher) are used update rates of 400 microseconds are possible

1. Using the Master/Slave functions simplifies linear moves. It is possible to have two master and two slaves working independently of each other. Any axis can be master or slave. Assuming a linear move of two axes should take place program the following
   a. Master = axis 0, Slave1 = axis1
   b. Use the following functions:
   c. SetProfileMode(slaveAxis, electronicgear) //sets the profile for this axis to electronic gear
   d. SetGearMaster(axis,masterAxis,source) //axis=slave axis, masterAxis=master axis, source=actual position or commanded position
   e. SetGearRatio(slaveAxis,ratio) //sets the ratio between master and slave. Example: 32768 moves the slave 0.5 counts for every positive master count.
   f. Update(slaveAxis) //updates the data
   g. Now any movement of the master axis will result in a proportionate (gear ratio) move of the slave axis or axes. The ratio is scaled by 1/65536 and can be positive or negative. The ratio can be changed anytime on the fly. A gear ratio of 1,000,000 would cause the slave to move 15.295 positive slave counts for every master count.
   h. The gear ratio is a buffered command and will not take effect until an update is issued. The ratio can be changed anytime on the fly, therefore great caution must be used.

- Using the On-board RAM provides the greatest flexibility. It allows execution of very complex profiles which must be precalculated by the host and then loaded into the RAM. The best way to explain the method is by an example. The files of the example ExternalProfile show how to program this way in Delphi. An example in C is also available. This could easily be translated to Visual Basic since this program makes use of the POSYS700.DLL (for the PCI version it would be the POSYS900.dll). This allows continuous path contouring for linear and circular motion. It also allows linear and S-curve acceleration and deceleration. It also gives the programmer flexibility in assigning update rates which can vary over the length of the program. The program length is practically unlimited since the RAM can be refilled with new data as stored data is being executed.
- The optional PathFinder development software provides the ultimate in ease of programming using the on-board RAM. It includes functions for complex move programming of linear and circular interpolation for continuous path motion with linear or S-curve acceleration/deceleration. The software also includes a G-code and HPGL interpreter. It is available as a DLL or in complete source (C/C++). The PathFinder user s guide is included for more detailed information.

Date: 23 July, 2002
Last Revision:

E-mail: support@servo-halbeck.com

¤

Digital IO Specification POSYS® 7xx/8xx/8xx-B/9xx

Digital IOs Specifications

Subject: POSYS® Series, Digital Input/Output Specifications

The POSYS® motion controllers are equipped with programmable digital IO. This TN provides details about the IO specifications for the POSYS® 700, POSYS® 800 and the POSYS® 900 series.

Type of IO	POSYS® 700	POSYS® 800	POSYS® 800-B/900	
Digital Inputs	10k Pullup to 5VDC	10k Pullup to 5VDC	10k Pullup to 5VDC , 74ACT541MTC	
Limit Switch Inputs	10k Pullup to 5VDC, MC14490DW	10k Pullup to 5VDC, MC14490DW	10k Pullup to 5VDC, MC14490DW	
AxisIn Inputs	4.7k Pullup to 5VDC	4.7k Pullup to 5VDC	4.7k Pullup to 5VDC , 74ACT541MTC	
Home Inputs	4.7k Pullup to 5VDC	4.7k Pullup to 5VDC	800-B	900
			4.7k Pullup to 5VDC,	
Encoder Inputs	Line Receiver SN75173, 150 Ohm terminating resistor	Line Receiver SN75173, 150 Ohm terminating resistor	Line Receiver DS26LS32C M, 150 Ohm terminating resistor	
Digital Outputs	TTL, 74FCT244, sink current: 48 mA, source current: 12 mA	TTL, 74ACT244, source/sink current: ±24 mA	TTL, 74ACT541, source/sink current: ±24 mA (max. ±50)	
Amplifier Enable Outputs	TTL, 74FCT244, sink current: 48 mA, source current: 12 mA	TTL, 74ACT244, source/sink current: ±24 mA	TTL, 74ACT541, source/sink current: ±24 mA (max. ±50)	
AxisOut Outputs	TTL, 74FCT244, sink current: 48 mA, source current: 12 mA	TTL, 74ACT244, source/sink current: ±24 mA	TTL, 74ACT541, source/sink current: ±24 mA (max. ±50)	

Date: 24.06.2003
Revision:

E-mail: support@halbeck.com

POSYS® is a registered trade mark of POSYS Motion Control

¤

Digital Inputs and Outputs

Subject: POSYS® Series, Digital Inputs and Outputs

The POSYS® motion controllers provide a number of digital inputs and outputs which ma be used in different ways.

1. Inputs:

    1. Encoder SignalsA+/A-, B+/B-, I+/I-. The signals A-, B- and I- are used in addition to the + signals if the encoder provides inverted signals (line driver) outputs. A+, B+ and I+ are the only signals used with TTL output encoders. Encoder input signals are dedicated inputs and cannot be used for other purposes.

    2. Limit Switchinputs XPLIM1, XNLIM1, XPLIM2, XNLIM2, XPLIM3, XNLIM3, XPLIM4 and XNLIM4 are used to monitor axis movement beyond a certain positive or negative distance. If the limit switch function has been activated by software, motion will stop if the appropriate positive or negative input goes active. It is possible to determine by software if the input should be active on a high or a low input signal. If the limit switches are not used to inhibit motion, it is possible to use these inputs as general purpose inputs. The state of the limit switch inputs (2 per axis) can be interrogated using the GetActivityStatus command. Bits 11 and 12 are set for an active input.

    3. AxisIninputs. There is one axis specific input per axis. These inputs can be used as dedicated inputs to trigger automatic events. These events can be a motion change (stop, start, change of velocity, etc.). The event is triggered upon a signal transition using the breakpoint mechanism. The AxisIn inputs can also be used as general purpose inputs. The command GetSignalStatus returns the state of the AxisIn input.

    4. Homeinputs are used for high speed position capture. When this input, one per axis, goes active the current actual position will be saved to a register of the motion control processor. The capture source must be defined by software since the encoder index can also be used to capture the position. The command to set the capture source is SetCaptureSource. The command GetCaptureValue returns the position value captured when a Home input went active. When used for high speed position capture, the inputs are axis specific. These 4 inputs can also be used as general purpose inputs. The command GetAxis_In_Out_Home returns the status of the Home inputs, the AxisIn inputs and the AxisOut outputs. The lower 4 bits (0..3) report the status of the 4 Home inputs, bits 4..7 report the status of the AxisIn inputs and bits 8..11 report the status of the AxisOut outputs.

    5. Uncommitted Inputs. There are 8 uncommitted digital inputs. Their normal sate is high (5 VDC) with pullup resistors of 10k. These signals are available to signal events to the operating program.

2. Outputs

    1. AxisOut, there is one AxisOut output per axis. These outputs can be programmed to track any of the assigned bits in ActivityStatus, EventStatus or SignalStatus register. The tracked bit in one of the three registers may be in the same axis or a different axis as the axis of the AxisOut pin itself. If the AxisOut outputs are not used for the above purpose it can be configured as a user programmable output. To do this select 0 as the register ID code in the SetAxisOutSource command and by adjusting the level of the resulting inactive output state to high or low as desired using the SetSignalSense command.

    2. AtRest output signals. These 4 outputs (one per axis) are only available with the stepping motor version (Pulse + Direction). The AtRest signal indicates when the trajectory generator is in motion. This signal is useful for stepper drivers that support different torque output levels for the motor in motion and for the motor at rest. No software setup is required to use these outputs. These are dedicated outputs and cannot be user programmed.

    3. AmpEnable outputs are provided to enable or disable attached servo amplifiers or stepping motor drivers. There are 4 outputs, one per axis. The state of these outputs after power-on reset and executing other reset functions like External Reset or a Hard Reset is LOW. These outputs can be set and reset by the software commands SetAmpEnable and SetAmpDisable. The command GetAmpEnable returns the status of all 4 outputs. If a reversed output level for enable and disable is required (HIGH level after power-on) inquire about our firmware upgrade. An in-field upgrade is possible. If these outputs are not required to enable or disable amplifiers or drivers it is possible to use them as user programmable outputs. The 3 commands are:

    4. SetAmpEnable(hPosys, mask) turns amplifier outputs on

    5. SetAmpDisable(hPosys, mask) turn amplifier outputs off

6. mask = GetAmpEnable(hPosys) returns the amplifier status

7. mask is a bit mask for the 4 outputs (1+2+4+8 for outputs 1,2,3 + 4)

8. hPosys is the card handle returned by OpenDevice(device)

9. User Outputs. There are 8 user programmable outputs. The power-on and after Reset state of these outputs is LOW.

10. PULSE + DIRECTION, these outputs apply to the stepping motor version only. The default behavior is that a step or pulse is considered to have occurred when the signal transitions from a high to a low output value. A high level of the direction signal indicates a positive direction pulse and a low level indicates a negative direction pulse. The logic for the pulse output can be inverted using the command SetSignalSense

For additional details about the mentioned commands please refer to the manual.

Example for AxisOut pin use

Assume the AxisOut pin should signal if the motor axis 0 (X) is in motion:

axis = 0
sourceAxis = 0
bit = 10
register = 2 (ActivityStatus)

SetAxisOutSource(axis, sourceAxis, register)

The AxisOut pin for the X-axis will now follow the status of bit 10 in the ActivityStatus register. The output will be set (high level) while a profile is being executed for the X-Axis. The level will revert to a low state when in position. Using SetMotionCompleteMode will specify if the bit should follow a commanded or actual position profile.

Date: 02.05.2003
Revision: 26.05.2004 – AmpEnable description revised

E-mail: support@halbeck.com

POSYS® is a registered trade mark of POSYS Motion Control

¤

Breakpoints

Subject: POSYS® Series, DLL BreakPoint Functions

All POSYS® series controllers support the DLL breakpoint functions. Breakpoints provide a convenient way of having the motion controller react to certain events. Each POSYS® axis has two distinct breakpoints that may be programmed for it. They are known as breakpoint 1 and breakpoint 2. Therefore it is possible to monitor two separate events for each axis.

This note describes how to program the breakpoint functions and uses an example to explain the steps required.

The following functions apply to the use of breakpoints:

Function	Description
SetBreakpointValue(axis, breakpoint_no, trigger_value)	This function sets breakpoint 1 or 2 for the selected axi to react to trigger_value
GetBreakpointValue(axis,breakpoint_no)	This function returns a long integer with the programme trigger_value
SetBreakpoint(axis, breakpoint_no, source_axis, action trigger)	This function establishes a breakpoint for the specified axis to be triggered by an event or condition of the source_axis. The source_axis may be the same or different then axis
GetBreakpoint(axis,breakpoint_no)	This function returns programmed source_axis, action and trigger
SetAxisOutSource(axis, source_axis, bit, register)	This function is not directly related to breakpoints but offers a convenient means of setting an out put without host intervention
GetAxisOutSource(axis)	This function returns the programmed values of source_axis, bit and register

For more details of the various commands please refer to the POSYS® manual.

Note:Always program SetBreakpointValue() first. This is very important since the motion controller will start to react immediately to a breakpoint event or condition once SetBreakpoint() is issued.

The following example demonstrates the steps required for using breakpoints based on an actual requirement.

Assume the following requirements:

1. X-motor starts to move towards a target position
2. Solenoid # 1 turns on after X-motor has reached ¼ of the target position
3. Y-motor starts to move to its target position after X-motor has traveled ½ towards the target position
4. Solenoid # 2 turns on when Y-motor is at ¾ of its target position
5. Solenoid # 1 and solenoid # 2 are turned on and off by axis specific outputs OUT1 (X-axis) and OUT2 (Y-axis)

Program Sequence, all values are only taken as an example:

All axes are assumed to start from absolute position 0.

```
Const
Xaxis = 0
Yaxis = 1
Speed = 01A36E (4000 steps/sec.)
Accel = 00DB (20000 steps(sec²)
TargetX = 20000 (quadrature counts)
TargetY = 34000 (quadrature counts)
Breakpoint1 = 0
Breakpoint2 = 1

Begin

SetVelocity(Xaxis, Speed)
SetVelocity(Yaxis, Speed)
SetAcceleration(Xaxis, Accel)
```

```
SetAcceleration(Xaxis, Accel)
SetPosition(Xaxis, TargetX)
SetPosition(Yaxis, TargetY) //values are written to the registers but not yet executed

SetBreakpointValue(Xaxis, Breakpoint1, (TargetX / 4))

//the breakpoint value for Breakpoint1 and the X-axis has been set to TargetX = 20000 / 4 =5000

SetBreakpointValue(Yaxis, Breakpoint1, (TargetX /2))

//set the Breakpoint1 of the Y-axis to the position of TargetX //= 20000 / 2= 10000

SetBreakpointValue(Yaxis, Breakpoint2, (TargetY * 0.75)

//set the Breakpoint2 of the Y-axis to the position of TargetY //= 34000 * 0.75= 25500

ResetEventStatus(Xaxis, GetEventStatus(Xaxis) and not 04H)

ResetEventStatus(Yaxis, GetEventStatus(Yaxis) and not 04004H)

//these commands clear the Breakpoint1 bits in the //EventStatusRegister of the X- and Y-axes if they were set, a //zero
(0) clears the bit. Breakpoint2 bit of the Y-axis is also //cleared

SetAxisOutSource(Xaxis, Xaxis, 2, 1)
//source_axis = Xaxis, Output OUT1 will be triggered when Breakpoint1 changes state, bit number = 2, status register = 1
(event). The OUT1 pin will follow the state of Breakpoint1 in the event register

SetAxisOutSource(Yaxis, Yaxis, 0E, 1)
//source_axis = Yaxis, Output OUT2 will be triggered when Breakpoint2 changes state, bit number = 0E, status register =
1 (event). The OUT2 pin will follow the state of Breakpoint2 in the event register

SetBreakpoint(Xaxis, Breakpoint1, Xaxis, 0, 1)

//this function activates the X-axis Breakpoint1 mechanism, source_axis = Xaxis, action = 0 (none), trigger = 1
(commanded position)

SetBreakpoint(Yaxis, Breakpoint1, Xaxis, 1, 1)

//this function activates the Y-axis Breakpoint1 mechanism, source_axis = Xaxis, action = 0, trigger = 1 (commanded
position). Note that the source axis is the X-axis! Action = 1 = Update starts the move of the Y-axis, trigger = 1 is the
commanded position of the X-axis

SetBreakpoint(Yaxis, Breakpoint2, Yaxis, 0, 1)

//this function activates the Y-axis Breakpoint2 mechanism, source_axis = Yaxis, action = 0 (none), trigger = 1
(commanded position)

Update(Xaxis) //this starts motion of the X-axis

End
```

Explanation of the program

The POSYS® axis specific outputs (OUT1 and OUT2) and its breakpoint mechanism are used to solve this requirement without the need of host intervention or polling.

After initializing the X- and Y-axes with speed, acceleration and target positions, the breakpoints and the source for the axis outputs (OUT1, OUT2) are setup.

Two breakpoints are available for each axis. Using the command SetAxisOutSource it is possible to set the event to which the outputs should react. The Breakpoint1 bits of the X and Y-axes in the GetEventStatus register will be set once the event/condition occurs as programmed in SetBreakpointValue and SetBreakpoint.

IMPORTANT

When programming SetBreakpointValue for conditions of EventStatus, ActivityStatus or SignalStatus, a two word mask has to be sent to the controller. The high word will be the mask for the trigger condition and the low word will be the mask for the sense (high or low) of the trigger bits. An example will best explain this requirement. Assuming for the EventStatus

register breakpoint number 2 should be triggered when the motion complete bit is set to one, bit 14 and bit 0 would be set. The value to be sent would be high_word=0x4001 and low_word=0x4001, so the value to be sent would be value = ((high_word shl 16) or low_word) = 0x40014001. In another example for the EventStatus assume a breakpoint should be recognized if the motion complete bit (bit 0) is set to 1 (motion complete), and the commutation error bit(bit 11) is set to 0. In this case high_word = 0x0801 and the low_word = 0x0001. The command to be sent actually would be the value = ((high_word shl 16) or low_word) = 0x08010001

SetBreakpointValue(axis, breakpointNo, 0x08010001)

Explanation: shl = shift left (in C/C++ <<)

or = or (in C/C++ |)

In order to reset OUT1 and OUT2 it is necessary to reset the corresponding bits in the EventStatus register. The ResetEventStatus will do just that. To reset OUT1 issue the following command

ResetEventStatus(Xaxis, GetEventStatus(Xaxis) and not 04)

Any zeros in the bit mask will reset those bits. In the above command, the Event Status register is read in with GetEventStatus(Xaxis) and bit 2 is set to zero and reset. Output OUT1 will revert to its previous state.

It is possible to change the sense state of the outputs OUT1 to OUT4 using the command

SetSignalSense(axis, mask)

A zero for the sense bit, the input/outputs is active low, a one for the sense bit makes the input/output active high. The OUT-bit is number 10, the encoding would be 0400H.

For example if the signal sense for the output OUT should be high, do the following

SetSignalSense(Xaxis, (GetSignalSense(Xaxis) or 0400H))

This turns on bit 10 in the mask.

Another interesting aspect is the use of interrupts with breakpoints. Breakpoint1 and Breakpoint2 can both trigger an interrupt. To activate this condition use the command

SetInterruptMask(axis, interruptMask)

To generate an interrupt on Breakpoint1 and Breakpoint2 bit 2 and bit 14 must be set to 1. The corresponding Event Status register bit will cause an interrupt when the status register bit goes active. To cause an interrupt on the X-axis's Breakpoint1 send the following:

SetInterruptMask(Xaxis, GetInterruptMask(Xaxis) or 04H)

After an interrupt has been serviced it must be cleared using

ClearInterrupt

and

ResetEventStatus(axis, GetEventStatus(axis) and not bit)

If interrupts are to be used it will be necessary to call an ISR (interrupt service routine) which will execute the desired routines. In this example it would be possible to have the ISR execute special host commands to a separate user I/O card. If no interrupt handler is installed it would be possible to use polling. The command

GetProcessorStatus

is a special instruction that returns the motion processor status. It returns 16 bits with the following meaning:

Bit number Description

0 – 12 unused, set to 0

13 holds value of HostIOError signal

14 holds value of Host Interrupt Signal, a 1 indicates a high level

15 holds value of Host Ready Signal, a 1 indicates a high level

Date: 06.09.2002

Revision: 15.01.2003

E-mail: support@halbeck.com

POSYS® is a registered trademark of POSYS Motion Control

¤

Error Code

Subject: POSYS® Series, Error Code

For more details of the various commands please refer to the POSYS® manual.

The POSYS® motion controllers can be interrogated to determine if an error occurred.

Two functions are useful for this. They are:

GetProcessorStatusand

GetHostIOError

GetProcessorStatus is a status read function that can be called anytime without a wait time. There is no check of the processors' ready bit.

The table defines the returned signal bits.

Bit Number	Description
0-12	Unused, set to 0
13	Holds value of HostIOError signal, refer to GetHostIOError function
14	Holds value of HostIntrpt signal, a 1 indicates the signal level is high
15	Holds value of HostRdy signal, a 1 indicates the signal level is high

If a call to GetProcessorStatus returns with Bit 13 set (1) it indicates an error condition. If this is the case a call to GetHostIOError would return a code indicating the cause of the error. The following table lists all of the error conditions:

Code	Indication	Cause
0	No error	No error condition
1	Motion chip reset	Default value of error code on reset or power-up
2	Invalid instruction	Instruction is not valid in the current context, or an illegal instruction code has been detected
3	Invalid axis	The axis number contained in the upper bits of the instruction word i not supported by the chip set
4	Invalid parameter	The parameter value sent to motion processor was out of its acceptable range
5	Trace running	An instruction was sent that would change the state of the tracing mechanism while the trace is running. Instruction which can return t error areSetTraceVariable, SetTraceMode and SetTracePeriod
6	Reserved	
7	Block bound exceeded	1. The value sent bySetBufferLength, or SetBufferStartwould create a memory block which extends beyond the allowed limits 400h – 7FFFFFFF 2. Either SetBufferReadIndex, or SetBufferWriteIndexsent an index greater than or equal to the block length
8	Trace zero	SetTraceStartImmediatewas issued, but the length of trace buffer is currently set to zero
9	Bad checksum	Applies to serial port only! The checksum compiled and returned by motion processor does not match that sent by the host
Ah	Not primary port	A prohibited instruction (one which can be executed only through the primary port) was issued through the diagnostic port
Bh	Negative velocity	An attempt was made to set a negative velocity without the axis bein in velocity contouring profile mode
Ch	S-curve change	The axis is currently executing an S-curve profile move and an atten was made to change the profile parameters. This is not permitted
Dh	Limit event pending	A limit switch event occurred

Code	Indication	Cause
Eh	Move into limit	An attempt was made to execute a move without first clearing the lir bit(s) in the Event Status register

Executing the GetHostIOError command clears both the error code and the I/O error bit in the I/O status read word (GetProcessorStatus).

The GetHostIOError function call takes more time then a call to GetProcessorStatus. The reason is that GetHostIOError checks the motion processor's ready bit and GetProcessorStatus does not. The suggested approach for error checking is shown in the following example:

Section calling Motion Processor Functions

```
SetVelocity(axis, speed)
If (GetProcessorStatus and Bit13 = 2000h) then
GetErrorRoutine // only if bit 13 is set go to the GetErrorRoutine and call GetHostIOError to determine the error cause
```

Section retrieving the error code and handling of the error

```
GetErrorRoutine
Var
Errorcode
Begin

Errorcode = GetHostIOError //a call to this function will reset all error bits
if (Errorcode and 2) = 2 then
DisplayMessage //display a message or go to an error handling routine
HandleError
End
End
```

Date: 24.10.2002
Revision:

E-mail: support@halbeck.com

POSYS® is a registered trade mark of POSYS Motion Control

¤

On-board RAM

Subject: POSYS® Series, On-board RAM

All POSYS® Series controllers are equipped with on-board RAM which in this document will be referred to as on-board RAM. The POSYS® 700 and 800 Series controllers make 256k x 8 available and the POSYS® 900 series makes 512k x 8 available.

This RAM serves two purposes, one is to use it as a trace buffer, the other is to use for storing and executing complex motion profiles. Using on-board RAM for the trace buffer makes it possible to collect real time data generated by the motion processor and store it automatically in RAM. Four different variables can be selected from a total of 28. It is possible to trace 4 variable at a time, i.e. data can be captured for each variable per update period (servo cycle).

The length of the trace buffer is variable and programmable. It could be up to the full available memory minus 512 words for the system.

If memory is needed for program storage and execution, again it could be the full length minus 512 words or it could be setup to have trace buffers and profile buffers. The different buffers are identified by Id's, the trace buffer must always be assigned ID = 0.

Relevant On-board RAM Functions

Functions which are relevant to the use of the on-board RAM are:

Profile Mode

- GetBufferFunction
- GetBufferLength
- GetBufferReadIndex
- GetBufferStart
- GetBufferWriteIndex
- ReadBuffer
- SetBufferFunction
- SetBufferFunction
- SetBufferLength
- SetBufferReadIndex
- SetBufferStart
- SetBufferWriteIndex
- WriteBuffer

The function SetProfileMode(axis, mode) has been expanded to include the on-board RAM mode = 4.

All of the above functions and the required parameters are described in detail in the Programmer's manual.

Trace Mode

- GetTraceCount
- GetTraceMode
- GetTracePeriod
- GetTraceStart
- GetTraceStatus
- GetTraceStop
- GetTraceVariable
- SetTraceMode
- SetTracePeriod
- SetTraceStart

- SetTraceStop
- SetTraceVariable

These functions only apply to trace function setup and execution. A detailed description can be found in the Programmer's manual.

Use of On-board RAM for Profile Mode

The on-board RAM can be used to store and execute complex motion profiles. Motion data calculated by the host is loaded into RAM and then executed by the motion processor without further host intervention. Up to 4 axes can move on a complex path in continuous fashion. Each axis can be programmed for

- Position
- Velocity
- Acceleration
- Jerk (S-curve)
- SegmentTime

All data is in 32-bit format.

- Position data is always absolute and represents the target destination at the end of the segment.
- Velocity is in quadrature counts/update period (servo cycle)
- Acceleration is quadrature counts/update period^2
- Jerk is in quadrature counts/update period^3
- Segment time is a multiplier for the update period (servo cycle)

If 4 axes are enabled and the update time is the default of 408 microsecond then this is the fastest available update time for data in the buffers represented by a SegmentTime multiplier of 1. If a SegmentTime multiplier of 0 is encountered in the time buffer the external profile mode for that axis will be exited. The other axes will continue to execute their profile. Motion errors and an abrupt stop will also cause an exit from on-board RAM.

The length of the segment time depends on the type of profile and the application. The shorter that time the more memory will be required. Movement along linear segments should use much longer segment times than circular motion. On circular motion segment times of less than 2 milliseconds are hardly ever required.

What must be observed

When a motion profile in on-board RAM is executed it will eventual get to the end of the buffer with data which had been allocated. If it does not find a SegmentTime of 0 it will wrap around to the starting position and continue with the execution. This could be potentially dangerous. If the final axes positions are different from the profiles starting positions it could cause an instant jump to these positions. To exit the on-board RAM profile mode the last segment must contain a SegmentTime multiplier of 0 for each involved axis. The axes will then remain in a stationary mode. The selected profile mode (SetProfileMode(axis, profile)) is still External (profile = 4) and must be set to the desired mode for further execution.

Available Memory (RAM) not sufficient for program

The program length exceeds the available memory. This is no problem. Divide the motion sequences into shorter segments. Load these shorter segments and send additional data (segments) to the on-board RAM after memory becomes available. The memory is split in a part that is executing and the other for storing data to be executed. The functions GetBufferReadIndex and SetBufferWriteIndex are the mechanism to determine the momentary read position within the buffer to specify a new write position for new data. Once the original data gets to the end the read pointer wraps automatically to the start position and continues executing the program.

Assume the following conditions, the total number of program positions in RAM required is 26,350, but only 10,000 are available. Proceed as follows:

- Load data into position 0 to 10,000
- Start external profile mode
- Set up a breakpoint which corresponds to some event as the BufferReadIndex passes the RAM position 5000 or check the BufferReadIndex for >= 5000
- Start loading new data into RAM starting at the BufferWriteIndex = 0 up to RAM position = 4999
- Check for wrap around until BufferReadIndex <= 10000
- Start loading new data into RAM starting at the BufferWriteIndex = 5000 up to RAM position = 10000

Continue this method until all data is loaded and executed. Make sure to append one segment with a SegmentTime

multiplier of 0 to exit External Profile mode.

¤

Limit Switches

Refers to: all POSYS® Motion Controllers

All POSYS® series controllers support travel limit switches. These limit switches, there are two per axis, can be configured to stop motion beyond a certain position of the machine. The switch is activated as the machine tries to move beyond it. This is an important safety feature and should not be by-passed.

This note describes how to program the limit switch functions and what to do once a limit switch has been activated and the machine has stopped.

The following functions apply to the use of limit switches:

Function	Action
SetLimitSwitchMode	enables or disables the limit switch for the specified axis
GetLimitSwitchMode	returns the programmed mode
SetSignalSense	set selected limit switch to be active high or active low
GetSignalSense	returns the programmed or default values
GetEventStatus	returns the event status including limit switches
ResetEventStatus	resets selected event bits

The following example demonstrates the steps required for using limit switches for the X-axis.

```
Const
axis = 0; // X-Axis
on = 1;
off = 0;
```

Function	Action
SetLimitSwitchMode(axis, on);	Turns on limit switch sensing for the desired axis

Use SetSignalSense to program limit switch sensing to be active high or active low; the default is active low

If during a move a limit switch is activated, the motor will come to an abrupt stop by automatically setting the velocity to zero, no deceleration is used. If the positive limit switch was activated, then no further motion will be possible in the positive direction. Unless this was intended as a normal operation, make sure the cause of the event is investigated and appropriate safety corrections are made.

As the next step clear (reset) the limit switch bit(s) in the event status register by issuing the ResetEventStatus for the appropriate axis. For the example issue

Function	Action
ResetEventStatus(axis,0);	this command would clear all pending bits in the event status registe

If only the bit of the positive limit switch should be cleared issue

Function	Action
ResetEventStatus(axis, 0x48DF);	this only resets bit 5,'In Positive Limit 0 in the mask resets the event, a value of 1 has no affect

Now a move should be made in a direction opposite to the one that caused the limit switch event, in this case in the negative direction. Trying to move into the positive direction would re-trigger the limit switch event.

Function	Action
SetProfileMode(axis, trapezoidal);	other modes like velocity are also possible only required if a differen mode is desired from the active one when the limit switch was activated
SetVelocity(axis, speed);	a limit switch activation sets the speed to zero, so before a new mov

Function	Action
	is possible a new value must be programmed
SetPosition(axis, GetPosition(axis)-5000);	this command sequence would get the present absolute position and move the motor–5000 counts away from it. The move distance chose should be sufficient to get it off the limit switch
Update(axis)	this starts the actual move

If an occasion occurs in which both limit switches (positive and negative) are active at the same time, for example both limit switch bits would be set in the activity status register, motion in either direction would be disabled. In this case use

Function
SetLimitSwitchMode(axis, off);

To disable limit switch processing temporarily until the motor is moved off the switches..

The limit switch inputs of the POSYS® motion controllers are equipped with digital filtering to avoid false triggering due to electrical noise.

When a limit switch event occurs the motor will always stop without a deceleration ramp. This is normally the desired action since the purpose of the limit switch is to prevent further travel. However, due to mechanical characteristics of some equipment (e.g. large inertia loads or heavy weights being moved) a smooth stop may be preferable.

This can be achieved using the breakpoint or interrupt system. In both cases the default action of the chip must be disabled. Here is a description of using a breakpoint, interrupts are similar but the SetInterruptMask must be set to SetInterruptMask(axis, bpNo).

Programming the BreakPoint system is done as follows:

Function	Action
SetLimitSwitchMode(axis, off);	This prevents the motion controller from setting the positive/negativ bits in the Activity- and Event Status registers and from stopping the motors

Using a BreakPoint to stop a motor:

SetBreakpointValue(axis, bpNo, (NEGLIM+POSLIMIT <<16) | (NEGLIM+POSLIM))
// << = shift left, | = or

SetBreakpoint(axis, bpNo, sourceAxis, action, trigger)
// bpNo = BreakPoint number 1 or 2
// sourceAxis = axis
// action = SmoothStop
// trigger = SignalStatus

Once a limit switch event occurs the motor will stop using the programmed deceleration for that axis.

For more details of the various commands please refer to the POSYS® manual.

IMPORTANT NOTE: This note applies to drivers without a handle. If the new POSYSPci drivers are used a handle must be included with every command, the handle is the first parameter to be passed as shown in the following example:

SetLimitSwitchMode(hPosys, axis, off)

hPosys is the handle (integer) returned that was returned when opening the driver:
hPosys = OpenDevice(deviceNo)

Date: 06.09.2002
Revision:

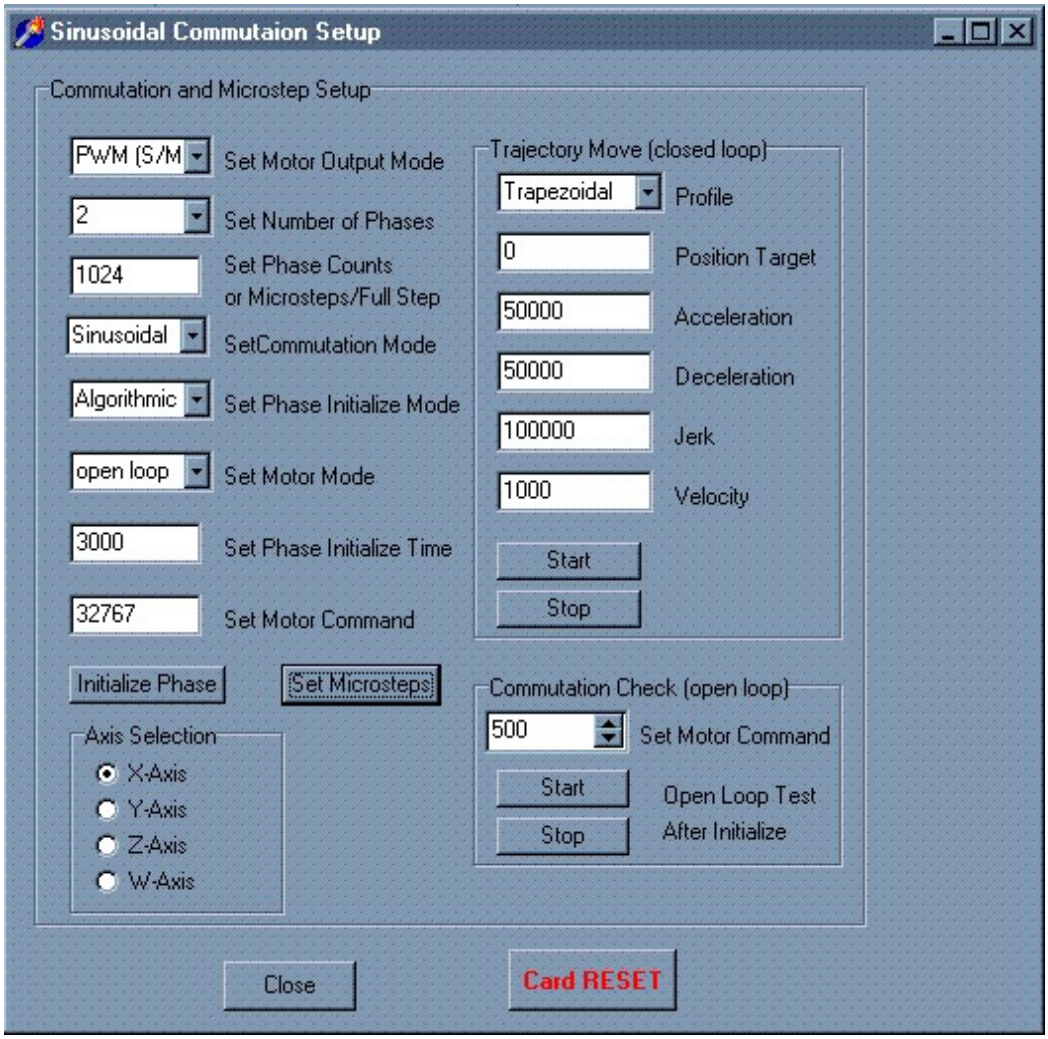
E-mail: support@halbeck.com

POSYS® is a registered trade mark of POSYS Motion Control

¤

Sinusoidal Commutation and Microstep Mode Setup

Refers to: POSYS® 84x-B/86x-B/87x-B/88x-B; 94x/96x/97x/98x; 182x; 192x



On the main menu select Commutation/Microstep / Setup

From the Axis Selection box click on the selected axis. For each axis you need to repeat this procedure

From the upper left combo box select the desired motor output mode (DAC, PWM (S/M), etc.

The next combo box serves for the selection of number of phases, select 2

Into the next edit field enter the number of microsteps/full step. For example for 64 microsteps enter 64 x 4 = 256, for 256 microsteps enter 256 x 4 = 1024 (this is the maximum)

The last edit field is used for limiting the output. You must enter a value larger than 0 or no output will happen. The value 32767 represents the maximum output.

No other entries are required for the Microstep (MC240000xx) motion controller.

Now click on the button Set Microsteps. This will load the parameters. Repeat the above steps for all axes that you want to use.

Now return to the Main Menu and go to the Stop On Motion Error selection box. Select the Radio Button Off.

If you selected for the Motor Output DAC it is very important to select in the ±10 Volt Outputs box the check button On, otherwise there will be no analog outputs.

You are now ready to enter move parameters, click on Send Data and Start Motion and your motors should start turning.

For general information the source that is executed when clicking on Set Microsteps is included:

```
procedure TForm10.Button7Click(Sender: Tobject);                    //Button Set Microsteps
var
ioerror :string[4];
index :integer;

begin
Imp900.GetHostIOError(hPosys);
index:=ComboBox1.ItemIndex;
if index=-1 then
index:=0;
Imp900.SetOutputMode(hPosys,SelAxis,index);
index:=ComboBox2.ItemIndex;
if index=-1 then
index:=0;
Imp900.SetNumberPhases(hPosys,SelAxis,index +2);                    //number of phases

Imp900.SetPhaseCounts(hPosys,SelAxis,StrToInt(Edit1.Text));         //phase counts

Imp900.SetMotorCommand(hPosys,SelAxis,StrToInt(Edit2.Text));        //max motor output
if (motioncontroller and $28000000)=$28000000 then

Imp900.SetCommutationMode(hPosys,SelAxis,2);                        //this command is only
                                                                    //valid and required for the
if (Imp900.GetProcessorStatus(hPosys) and $2000)=$2000 then begin   //MC280000xx chip
ioerror:=IntToHex(Imp900.GetHostIOError(hPosys),2);
ShowMessage('Host IO error: '+ioerror);
exit;
end;
end;
```

Rev. Date: 08.01.2004
added comments for ±10 Volt Outputs box

¤

Mixed Motor Mode

Subject: POSYS® 800-B and POSYS® 900 Series

The POSYS® 800-B and the POSYS® 900 Series motion controllers are identical with the exception of the bus system to the host. The POSYS® 800-B is a PC-104 bus based controller, the POSYS® 900 is a PCI bus based controller.

Both types can be operated in the so-called mixed motor mode. To use this mode the controllers must be a fully featured POSYS® 972/974 (including Synch POSYS® 982/984) or POSYS® 872-B/874-B (including Synch POSYS® 882-B/884-B). This mode allows the operation of any axis in the brushed servo mode, brushless servo mode, brushless servo mode with on-board sinusoidal commutation or in the microstepping mode. The axes can be mixed in any combination like for example X-axis = brushed, Y- and W-Axis = brushless (sinusoidal) and Z-Axis = microstepping.

When the system starts up or after a Reset, all axes wake-up in the brushless servo mode with number of phases set to 3 and the output mode is set to 2.

To set an axis to brushed, brushless or microstepping the following commands must be issued:

Command	Brushed Servo	Brushless Servo	Microstepping
SetNumberPhases(axis(see Note 1, phase	1	2 or 3 (see note 2)	2
SetCommutationMode(axis,mode)	n/a	0, 1 or 2	2
SetOutputMode(axis,mode)	0, 1 or 2	0, 1 or 2	0, 1 or 2
SetPhaseCounts(axis, counts)	n/a	see manual	(Microsteps/Full Step) x 4

Note 1: Axis refers to the addressed axis 0, 1, 2 or 3 (X, Y, Z or W)

Note 2: PWM Sign/Magnitude only possible with number of phases set to 1 or 2.
PWM 5050 output mode is possible with number of phases set to 1, 2 or 3.

When using the controller in the brushless mode with on-board commutation it is important to observe the programming requirements for Sinusoidal Commutation.

An excellent tool to sinusoidal setup is the document PhaseInitialization.pdf with a step by step guide .

Date: 04.12.2003
Last Rev.: POSYS Motion Control GmbH & Co. KG

Tel: +49 (0)9158-1612
Fax: +49 (0)9158-1642
www.halbeck.com

¤

Multiple Cards (only PCI)

Subject: POSYS® Series, PCI bus, multiple cards

This TN describes the usage of the new PCI drivers for single and multiple cards installed in the same computer. The new driver requires a handle to identify the addressed card. This also requires the new POSYS900 DLL, version 3.0 and higher.

Instructions

Install the POSYS® 900 motion control cards in the PC PCI slots. The driver will support a maximum of 16 cards (0 to 15). Now restart the computer. The newly installed POSYS® motion controllers will be detected and you will be asked for the directory of the drivers.

Locate the directory containing the drivers and the POSYS900.INF file.

It should contain the following files: POSYS900.INF

POSYSPCI.DLL
POSYSPCI.SYS
POSYS900.DLL

These files will be automatically installed in the correct directories of Windows 98/ME or 2000/XP. Depending on the operating system you may be asked to restart your computer.

It is now possible to communicate with all installed POSYS® cards. Here is a list of the applicable commands (Delphi).

```
Function DetectPosysDevices:word;stdcall;
//returns the number of installed cards, 0 if none were found

Function OpenDevice(device:byte):Thandle;stdcall;
//returns the handle of device, device starts at 0 for first card, 1 for second ,etc.
//OpenDevice returns –1 if the driver could not be opened

Procedure CloseDevice(hPosys:Thandle);stdcall;
//The device should be closed at the end of the program
```

Example:

```
Type
Thandle = integer;
Var
hPosys0,
hPosys1 :Thandle

if DetectPosysDevices= 0 then begin
ShowMessage('No Motion Controller found on PCI-bus!');

Close;

end;

hPosy0 := OpenDevice(0); //opens the driver and returns the handle of card 1

if hPosys0 = -1 then begin
ShowMessage('Driver could not be started!');

Close;

end;

InitializePOSYS(0,0);
hPosys1 := OpenDevice(1) //opens the driver and returns the handle of card 2
if hPosys1 = -1 then begin
ShowMessage('Driver could not be started!');
```

```
Close;

end;

//If there are more cards, then continue with hPosysx := OpenDevice(x)
```

It is now possible to communicate with the cards, the card is chosen by passing the handle to the functions and procedures in addition to the addressed axis and the values.

Example:

```
SetPosition(hPosys0, axis, position)
//sets the target position of card 1 with the hPosys0 handle

SetPosition(hPosys1, axis, position)
//sets the target position of card 2 with the hPosys1 handle
//all other commands are called in the same manner

CloseDevice(hPosys0) //close instance for card 1
CloseDevice(hPosys1) //close instance for card 2
//Before terminating the program all instances of opened drivers should be closed
```

Date: 20.04.2004
Revision: 1.0

E-mail: support@halbeck.com

POSYS® is a registered trade mark of POSYS Motion Control GmbH & Co.KG

¤

PCI ScanForPOSYS

Subject: POSYS® 900/1900 Series only , PCI Scan

This Technical Note applies to the POSYS® 900/1900 series motion controllers only.

The addresses and interrupt number of PCI devices are assigned by the computer's BIOS during start up. The user cannot assign addresses to the card like customary and required for ISA bus cards. The POSYS® 900 PCI bus cards do not provide any DIP-switch or jumpers for address and interrupt setting. As long as the card is not changed to a different slot the address and interrupt number assignment will not change from one power on to the next.

The POSYS900.dll library makes available a function that will scan the PCI bus and detect the first POSYS® 900 series motion controller on the bus.

```
var
CS0, CS1 : word;
success : Boolean;
```

DLL Function: GetAddresses(CS0,CS1,success)

Returns: Base Address 1 in CS0 and Base Address2 in CS1

success = true if a POSYS® 900 series motion controller was found
success = false if no PSYS 900 series motion controller was located

To assign all used addresses a call to

InitializePOSYS(CS0, CS1)

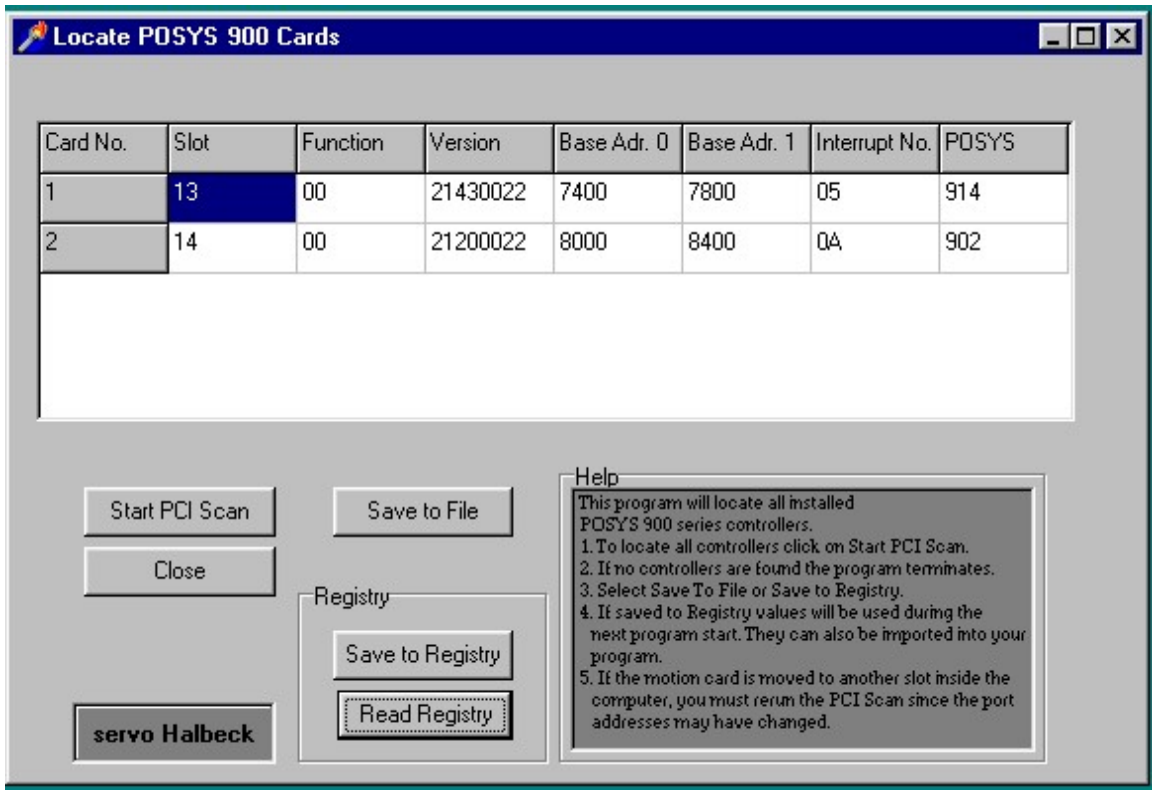
(actually this call could be skipped since it is already called by GetAddresses if a POSYS® 900 controller was located) is required. After these two calls all POSYS900.dll functions can be used.

If no PCI scan is desired after every program start, or if more than one card are installed, it is necessary to run the utility

FindPOSYS900

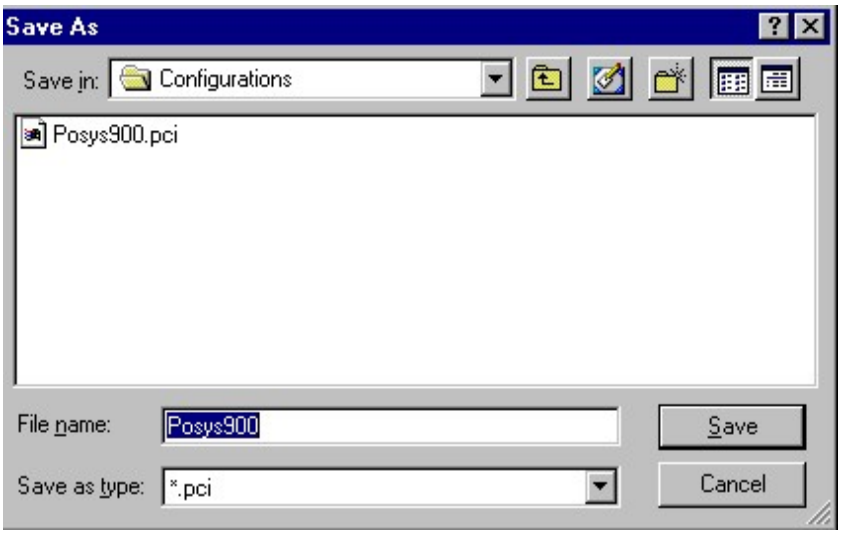
A window will open up and after clicking on the Start PCI Scan button the following information will be displayed.
In this example, Card 1, a 4 axes card (POSYS® 914) was located at Base Address 0 =7400H and Base Address 1=7800H and Interrupt No. 05H.

Card2, a 2 axes card (POSYS® 902) was located at Base Address 0=8000H and Base Address 1=8400H and Interrupt No. 0AH

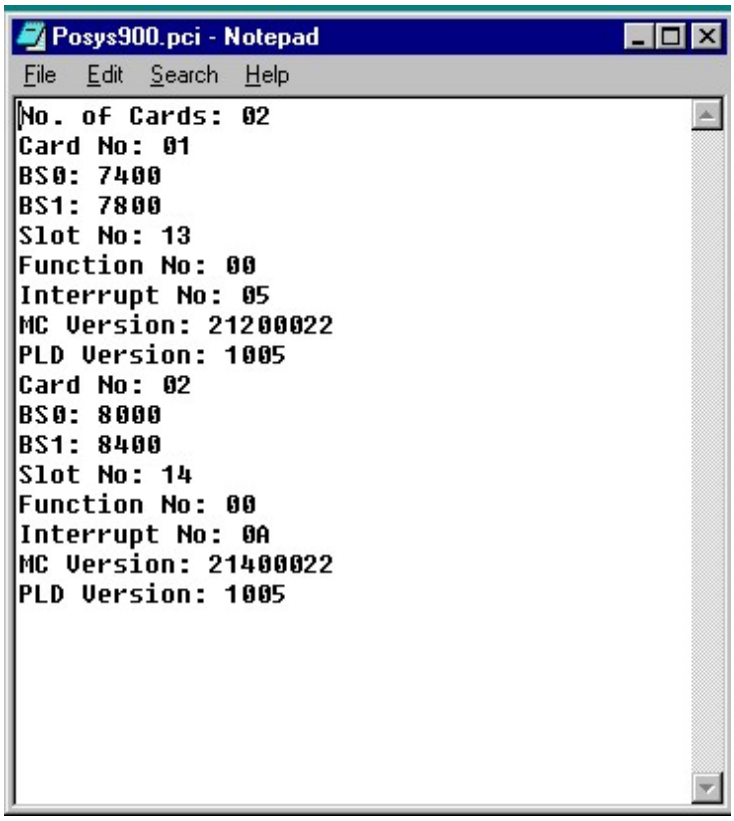


ScanForPosys Window

Four options are available. Click on the Close button and exit the program without saving any information. After clicking on Save to File the following dialog box opens up allowing selection of directory and name of the file for saving.



The next window displays the information saved in the file Posys900.pci:



The first line reports the number of POSYS900 PCI motion controllers found during the scan, in this case 2. The second line starts with information about Card 1. Information is stored for:

- BS0: Assigned Base Address 0
- BS1: Assigned Base Address 1
- Slot No.: Slot Number of the card
- Function No.: Function Number of the card
- Interrupt No.: Assigned Interrupt Number of the card
- MC Version: Motion Control version and revision
- PLD Version: Version number of the FPGA

Card No2: This line starts information for Card 2

A user program can use this information after a program start to initialize the card of his choice or initialize all cards installed. To do this the user program must open the file and extract base address and interrupt information.

Clicking on the Save to Registry button will save the same information to the following registry locations:

Card 1:

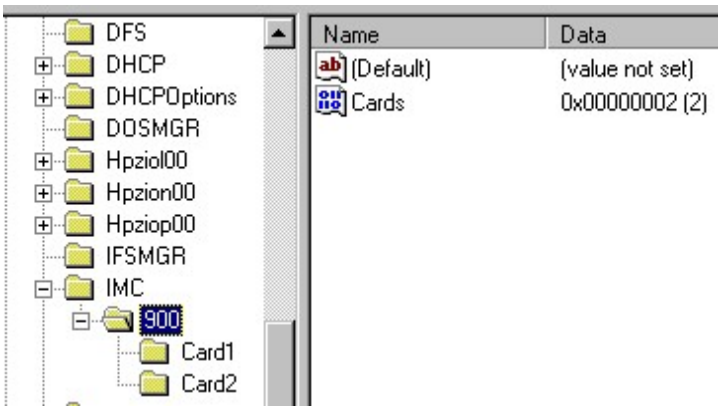
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\VxD\IMC\900\Card1

and

Card 2:

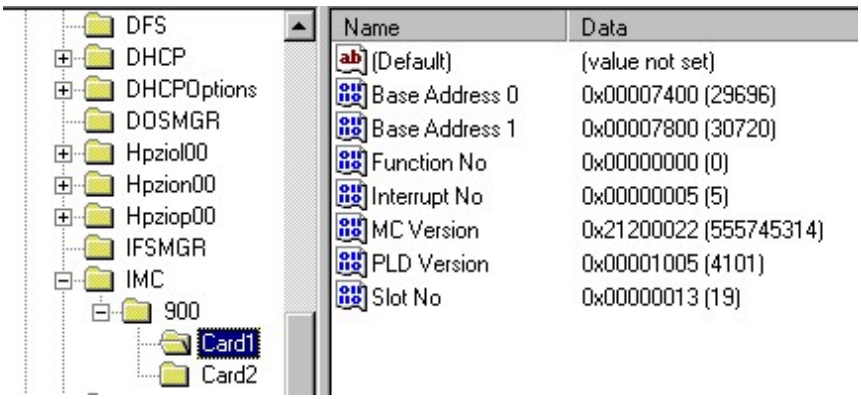
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\VxD\IMC\900\Card2

Additional cards would be located at Card3 and Card4.

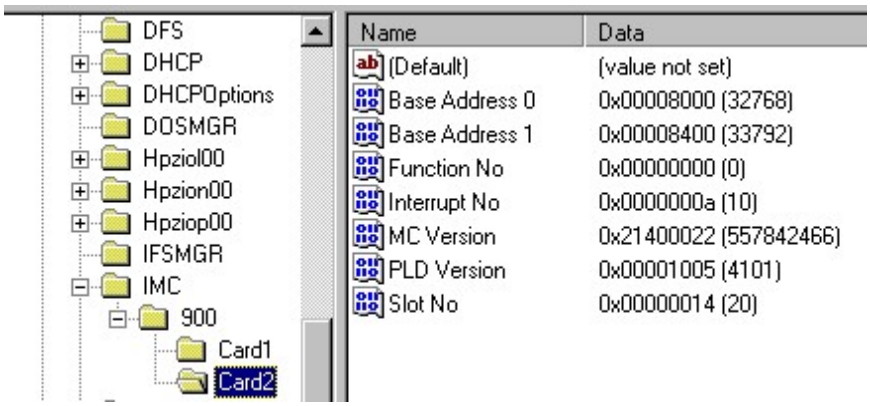


The number of cards found is written to the key Cards in location ..\IMC\900.

The values for each card are written to the various keys in locations ..\IMC\900\Card1



and to ..\IMC\900\Card2



The user program can extract this information from the registry and initialize the POSYS® 900 series motion controllers using the Base Address 0, Base Address 1 and Interrupt No. data.

The fourth option ‘Read Registry’ will return and display data that was stored in the registry. If no information was found the user will be advised and asked to run ‘Start PCI Scan’ first.

Important

If cards are moved from one slot to another the PCI scan must be repeated and the data saved again to file or registry.

The Tech Note ‘Programmers Info’ provides additional information for writing user programs in any language.

Date: 07.11.2002

Revision: 19.12.2002

E-mail: support@halbeck.com

POSYS® is a registered trade mark of POSYS Motion Control

¤

Programmer's Info

POSYS® 700/800/800-B/1800 Series

Please refer to paragraph 2 for POSYS® 900/1900 specific information.

This information is provided for programmers who want to write their own motion control program in a language of their choice and operating system of their choice (DOS, Windows, Linux, etc.).

Items needed:

Motion Processor Programmer's Reference
Motion Processor User's Guide

The newest versions of these manuals can be downloaded from our web site: http://www.POSYS Motion Control.com

The base address for the POSYS® 700/800/ is set using dip switches (description in the corresponding hardware manual). The card occupies N+7 address spaces (bytes), where N is the selected base address.

Example: Base Address N = 300H, so the card uses 300H to 307H. The address for a second card could start at 308H.

Name	Address setting 7xx/8xx	Address setting 8xx-B
Base Address	N	N
Data Address	N	N
Command Address	N + 2	N + 2
DACOn	N + 4	N + 4
DACOff	N + 5	N + 6
HardReset	N + 6	N + 5 and N + 7 (Byte)

Example:

Name	Address setting 7xx/8xx	Address setting 8xx-B
Base Address	300H	300H
Data Address	300H	300H
Command Address	302H	302H
DACOn	304H	304H
DACOff	305H	306H
HardReset	306H	305H

To communicate with the card these addresses must be assigned before writing or reading data.

Example:

The example functions are written in Delphi (Pascal) but could easily be translated to C.

```
Procedure InitializePOSYS(BaseAddress:word);
begin
if (BaseAddress < 512) or (BaseAddress > 1016) then // outside this range not recommended
exit;
DataAddress:=BaseAddress;
CommandAddress:=BaseAddress+2;
ResetAddress:=BaseAddress+6; (for 700/800)  :=BaseAddress + 5 (for 800-B);
DACOn:=BaseAddress+4;

DACOff:=BaseAddress+5; // works only with PLD versions 0392 (700 series) and 3104 (800 series)

DACOff:=BaseAddress+6; //works only with PLD versions 3205 (800B series)
```

```
end;
```

To read user inputs or write user outputs or amplifier enable outputs the commands ReadIO and WriteIO are used. To address the appropriate devices the correct IDs must be used.

Addresses for user IO:

Name	Address 700/800	Address 8xx-B	Description
amp_enable	02H	01H	WriteIO and ReadIO
write_user_output	00H	00H	only WriteIO
read_user_output	03H	00H	only ReadIO
user_input	00H	00H	only ReadIO
axis_in_out	01H	02H	only ReadIO
serial port	n/a	03H	Read/Write
watchdog	n/a	04H	only WriteIO
watchdog value	n/a	5562H	Value to write to watchdog

For an example let us assume your language command to write a word to the port is OUTPW(PortAddress, Value) and to read a word from the port is INPW(PortAddress). Reading and writing the port are motion processor commands requiring a check on the motion processors busy bit. Only if this bit is set (1) can communication take place. WaitIfBusy is an example for such a routine.

```
Const
readybit = $8000;
var
timeout : boolean;
count : word;
Procedure WaitIfBusy;
var
temp:word;
begin
count:=0;
timeout:=false;
repeat
inc(count); // increment count by one
temp := INPW(CommandAddress); // read one word from command address
until (((temp and $8000) = readybit) or (count = 2000));
if count = 2000 then timeout:=true; // make sure the loop ends if a timeout occurs
end;
```

Depending on the computer, operating system and programming language used the number of wait loops vary. On the average they will vary between 3 and 12.

Example:

Coding required to read the user inputs (8 lines)

```
Var
InputData : word;
WaitIfBusy;
OUTPW(CommandAddress,83H); // ReadIO
WaitIfBusy;
OUTPW(DataAddress,user_input);
WaitIfBusy;
InputData := INPW(DataAddress); //returns 16 bits (word)
```

This will return the status of the 8 input lines.

Example:

Coding required to write to the user output (8 lines)

```
OutputData := ffH (turn all 8 outputs on)
WaitIfBusy;
OUTPW(CommandAddress,82H); // WriteIO
WaitIfBusy;
```

```
OUTPW(DataAddress,write_user_output);
WaitIfBusy;
OUTPW(DataAddress,OutputData); // writes 16 bits (word) to DataAddress
```

Here are more examples reading and writing motion control data.

The first example shows how to read the encoder position (Command: GetActualPosition).

```
Function GetActualPosition(axis:word):longint; // axis = 0, 1, 2, or 3 for the 4 axes
var
temp1:word;
begin
temp1:=(axis shl 8) + $037; // shift axis left by 8 bits and add command code 037h
WaitIfBusy;
OUTPW(CommandAddress,temp1);
GetActualPosition:=GetDoubleWord; // reads the high and low order parts of the long
end;

Function GetDoubleWord:longint; // returns a 32 bit signed integer
var
temp:longint;
begin
WaitIfBusy;
with Long2Word(temp) do begin
wordHi:=INPW(DataAddress);
WaitIfBusy;
wordLo:=INPW(DataAddress);
end;
GetDoubleWord:=temp;
end;

Type
Long2Word = Record
wordLo, wordHi:word;
End;
```

The next example shows the required steps to turn off servoing and output an analog voltage to the servo amplifier.

```
Var
axis, mode : word;
axis := 0 ;
mode := 0 ; // 0= off, 1 = on

SetMotorMode(axis, mode)
SetMotorCommand(axis, value) // value range: –32768 to 32767 for full ±10 volts
Update(axis) // moves the buffered command to working registers
```

Depending on the magnitude of value an analog voltage will be output to the amplifier. This is in the open loop mode, no servoing, so extreme caution should be used. The processor will capture the encoder position which could be read using GetActualPosition(axis).

The POSYS® card provides a safety feature, after power on or an external reset, the torque demand outputs (16 bits analog ±10 volts) are connected to ground using optical switches. They must be connected to the ±10 volt outputs by the software command DACOutputsOn before any voltage can be sent to the amplifier inputs.

The coding for this command is:

```
OUTPW(DACOn); // DACOn = N + 4 (BaseAddress + 4)

POSYS® 7xx/8xx series only:
OUTPW(DacOff);       // DacOff = N + 5 (BaseAddress + 5) turns the optical switches OFF
                     // DacOff works only with PLD versions 0392 (700 series) and 3104 (800 series)

POSYS® 8xx-B series only:
OUTPW(DacOff);       // DacOff = N + 6 (BaseAddress + 6) turns the optical switches OFF
                     // DacOff works only with PLD versions 3205 (800B series)
```

Do not use WaitIfBusy with these commands or with the HardReset command.

The HardReset command will do just that, it will do an actual hard reset of the hardware. After this command all

parameters must be reloaded just like after a power on.

The coding for this command is:

POSYS® 7xx/8xx series only:
OUTPW(HardReset); // HardReset = N + 6 (BaseAddress + 6);

POSYS® 8xx-B series only:
OUTPW(HardReset); // HardReset = N + 5 (BaseAddress + 5) or
// HardReset = N + 7 (BaseAddress + 7)

Do not use WaitIfBusy with this command.

Examples for setting and resetting the amplifier enable outputs:

700/800 Series

WriteIO(amp_enable, 0x00) // sets all 4 outputs to low level
WriteIO(amp_enable, 0x0f) // sets all 4 outputs to high level
WriteIO(amp_enable, 0x03) // sets outputs 1 and 2 to high level, 3 and 4 to low level

800B Series, applies also to the POSYS 900 series

The 800B series can set and reset each amplifier output individually or any combination.

WriteIO(amp_enable, 0xffff) // sets all 4 outputs to high level
WriteIO(amp_enable,0x0300) // resets outputs 1 and 2 to low level
WriteIO(amp_enable,0xff00) // resets outputs 1, 2, 3 and 4 to low level
WriteIO(amp_enable,0x0404) // sets output 3 to high level
WriteIO(amp_enable,0x0400) // resets output 3 to low level

ReadIO(amp_enable) // returns the status of the amplifier enable outputs. Bit 7 reflects the status of the DACOutputs (1 = ON, 0 = OFF)

A HardReset command would return all set outputs and amplifier enable outputs and DACOutputs to the OFF state.

Please make sure to read the manual's description referring to the Reset command. This is also the motion processors status after an external reset and power on. The user must then configure the processor using for example a configuration file to his requirements.

After assigning the correct addresses for the card the following should be done. This only serves as an example.

Command	Description
InitializePosys(BaseAddress)	this assigns the addresses based on BaseAddress
WriteIO(write_user_output, 0)	set all outputs to 0 or to some other value
SetOutputMode(axis,mode)	mode: 0=16 bit analog, 1=PWM Sign/Magnitude
SetKp(axis, value)	value: 0 to 32767, suggested default is 100
SetKd(axis, value)	value: 0 to 32767, suggested default is 800
SetKi(axis, value)	value: 0 to 32767, suggested default is 40
SetIntegrationLimit(axis, value)	value: 0 to 32767, suggested default is 2000
SetPositionErrorLimit(axis, value)	value: 0 to 2 147 483 647 counts, suggested value is 2000
SetAutoStopMode(axis, mode)	mode: 0 = disable, 1 = enable, suggested value is 1
SetProfileMode(axis, value)	value: 0 to 3, suggested 0 = trapezoidal move
SetAcceleration(axis, value)	value: 0 to 2 147 483 647, suggested value 26214
SetDeceleration(axis, value)	value: 0 to 2 147 483 647, suggested value 26214
SetVelocity(axis, value)	value: 0 to 2 147 483 647, suggested value 26214
Update(axis)	updates all parameters
DACOutputsOn	connects analog outputs to amplifiers coding is OUTPW(DAC On), see above for more detail

The motion processor and the card are now initialized and it is possible to send target positions and move to those after issuing an update command for that axis. Be aware that adjustments may have to be made in filter and other parameters. For safety reasons the SetAutoStopMode should be enabled.

Interrupts

Only the motion processor will generate interrupts. The available interrupt sources are listed for GetEventStatus. The interrupt source is selected using a 16 bit mask and passed to the motion processor using the command.

SetInterruptMask(axis, interruptMask)

After an interrupt has occurred the host must determine the source by reading GetEventStatus(axis) and process it. The host must then reset the bit that caused the interrupt using the command.

ResetEventStatus(axis,mask)// a 0 in the mask clears the corresponding bits

The host must then send to the motion processor the command.

ClearInterrupt

Coding is:

```
WaitIfBusy;
OUTPW(command) // command = ACh (decimal = 172)
```

to clear the interrupt line and re-enable interrupt processing. If another interrupt is pending the interrupt line will only be cleared momentarily and then reasserted.

There is another very POSYS® specific command, called GetAxis_In_Out_Home. It provides information on the 4 AxisIn inputs, the 4 AxisOut outputs, the 4 Home inputs and the status of the optical switches (16 bit analog voltage outputs).

GetAxis_In_Out_Home(axis_in_out) // axis_in_out = 1

The coding is as follows:

```
WaitIfBusy;
ReadIO(axis_in_out); // this function returns 16 bits
```

The bits reflect the following:

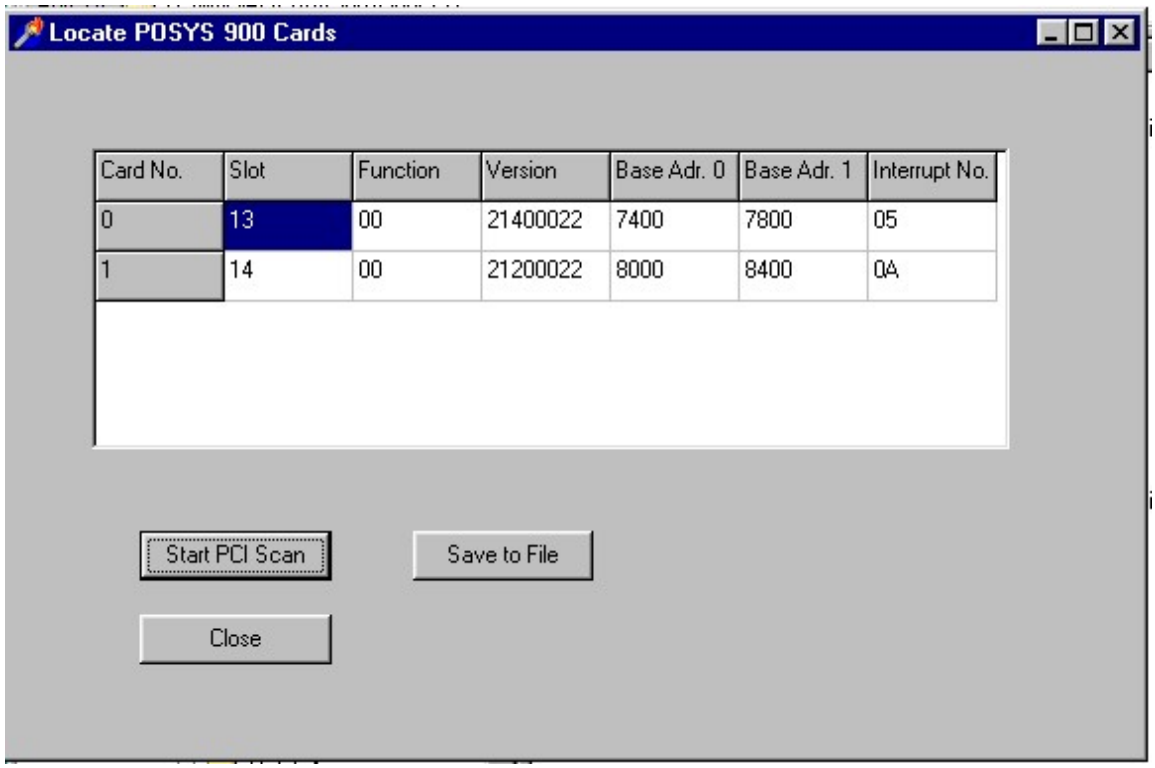
Bits 0 – 3	Status of Home inputs for axes 0- 3
Bits 4 - 7	Status of AxisIn inputs for axes 0- 3
Bits 8 – 11	Status of AxisOut outputs for axes 0- 3
Bit 12	Status of optical output switches (1 = enabled, 0 = disabled)
Bits 13 – 15	no meaning

POSYS® 900/1900 Series

The POSYS® 900/1900 motion controllers are for the PCI-bus and require no port address settings. These cards are Plug&Play compatible and require no DIP switch settings. If the system is used with the supplied drivers and DLL, the card will be automatically detected and addresses assigned. If experienced software engineers want to write programs close to the hardware then attention must be paid to the following:

Windows– for Windows 95/98, ME, NT, 2000 and XP use the supplied driver and follow the installation instruction.

Run the FindPCI program. It will locate installed POSYS® 900/1900 cards and display a window with information in regard to Card No., Slot, Function, Version, Base Address0, Base Address1 and Interrupt No. Version, Base Address and Interrupt Numbers are displayed in hexadecimal.



Results of a PCI scan

In the above example two POSYS® cards were located, a POSYS® 904 (Version 21400022) and a POSYS® 902 (Version 212000022).

Programmers can now use this information in writing their own drivers. The following addresses are used. Please note, these addresses deviate from the ones used for the POSYS® 700/800 series.

Assignment of addresses:

Name	Address
Base Address	Base Adr0
Data Address	Base Adr0
Command Address	Base Adr0 + 2
DACOn	Base Adr0 + 4
DACOff	Base Adr0 + 6
HardReset	Base Adr1

Example:

Name	Address
Base Address	7400H
Data Address	7400H
Command Address	7402H
DACOn	7404H
DACOff	7406H
HardReset	7800H

Addresses for user IO:

Name	Address	Description
amp_enable	01H	WriteIO and ReadIO
write_user_output	00H	only WriteIO

Name	Address	Description
read_user_output	00H	only ReadIO
user_input	00H	only ReadIO
axis_in_out	02H	only ReadIO
serial_port	03H	read/write the serial port
watchdog_adr	04H	write to watchdog register
watchdog_value	5562H	value to write to watchdog
pld_adr	FFH	address at which to read the PLD

To read back the status of user IO and to get the status of the inputs and outputs use

ReadIO(user_input) or you may use

ReadIO(read_user_output)

The return value is 16 bits, the lower 8 bits [0..7] reflect the status of the programmable user outputs, the upper 8 bits [8..15] reflect the status of the 8 input lines.

To set the outputs use

WriteIO(write_user_output, value)

This writes a word to the userIO. The lower 8 bits [0..7] will set the 8 outputs. The upper 8 bits are ‘don’t care’.

The amplifier enable outputs are set and reset using the following scheme:

WriteIO(amp_enable, value)

Sets the 4 amplifier enable outputs bits [0..3], bits [4..6] are ‘don’t care’, bit [7] reflects the DACOn/Off status, bits [8..1] must be set with change mask reflecting the bits [0..3], bits [12..14] are ‘don’t care’ and bit [15] is the change mask for DACOn/Off.

To obtain the status of the amplifier enable outputs use

ReadIO(amp_enable)

This will return the status of the amplifier enable outputs in bits [0..3, bits [4..6] are fixed at 0, bit [7] returns the DACOn/Off status and bits [8..15] are fixed at 0.

Example for Amp Enable:

Assume you want to enable output 1 and 3 (from low to high)

```
Var
amps: integer=0;
Begin
amps:= amps or $101 // output 1
amps:= amps or $404 // output 3
WriteIO(amp_enable, amps)
End
```

Now assume you want to disable output 3

```
Begin
amps:=amps and not $04 or $400
WriteIO(amp_enable, amps)
End
```

To obtain the status of the axis specific inputs, outputs and home status use

ReadIO(axis_in_out)

This returns the status of the 4 Home inputs in bits[0..3], the status of the 4 AxisIn inputs in bits [4..7], the status of the 4 AxisOut outputs in bits [8..11] and the Reset cause in

Bit [12] Reset Cause: Software
Bit [13] Reset Cause: Under voltage
Bit [14] Reset Cause: External
Bit [15] Reset Cause: Watchdog

A write to address axis_in_out WriteIO(axis_in_out, 0) will reset the reset cause flags.

To activate the watchdog timer write

WriteIO(watchdog_adr, watchdog_value)

The watchdog times out after 104 milliseconds. It must be re-triggered before timeout. A timeout causes a reset of the board and the watchdog would have to be reactivated. The watchdog can only be deactivated by a reset.

The Mode jumper default setting is 1-2 for limited use of the serial port. Only Get instructions and the SetBufferReadIndex are permitted. Setting the Mode Jumper to 2-3 allows the Full instruction set to be executed. In both cases a special serial cable is required from POSYS Motion Control.

A read of

ReadIO(serial_port)

Returns the default data transmission parameters and the slave address in bits [0..15]. Refer to the manual's SetSerialPortMode/GetSerialPortMode command description for an explanation.

New parameters can be written using

WriteIO(serial_port, value)

Value is the mask for the transmission parameters.

Status of out puts after power-on and a HardReset:

All outputs are low (0)

Amplifier enable outputs are low (0)

Axes specific outputs (AxisOut) are high (1)

The user's program must assign addresses to call the POSYS® functions. If the Kithara drivers are used these calls must be made to the POSYS900.dll functions:

A function assigning addresses could look like this Delphi example:

```
Procedure InitializePOSYS(BaseAddr0, BaseAddr1:word);
begin
DataAddress:=BaseAddr0;
CommandAddress:=BaseAddr0 + 2;
ResetAddress:=BaseAddr1;
DACOn:=BaseAddr0 + 4;
DACOff:=BaseAddr0 + 6;
end;
```

The next DLL-function calls open the driver and allow access to the port addresses specified:

OpenDriver

EnableIO(BaseAdr0, 16)
EnableIO(BaseAdr1, 16)

When shutting down the program call

CloseDevice

These are the only differences in programming for the POSYS® 700/800, 800-B and the POSYS® 900/1900 series motion controllers. For additional information refer back to paragraph 1 (POSYS® 700/800).

Date: 04 May 2001

Revised: 18 June 2001
Revised: 12 July 2002
Revised: 18 December 2002
Revised: 01 March 2005

POSYS Motion Control GmbH & Co. KG

E-mail: support@halbeck.com

¤

Resistor Networks for TTL or inverted Encoder Signals

Encoder Line Receiver and Terminating Resistors

All of the POSYS motion controllers use line receivers. They are equipped with 150 Ohm terminating resistors for RS 422 operation. The encoder signals are connected to the A+ and A- (B+, B- and I+, I-) inputs. The 150 Ohm network resistors are connected across these inputs.

POSYS® 700/800/800-B/1800/900/1900
RS1, RS2, RS3

POSYS® 1800-B
RS604, RS608, RS613

If single ended (TTL level) encoders are used, these network resistors should be removed. The resistor networks are socketed and can just be pulled out.

If mixed operation is desired with TTL level encoders and others with inverted signals some manual work is required. The 150 Ohm resistors on the TTL level inputs need to be selectively removed by clipping off the terminals. The table shows the resistor to encoder relationship.

RS1/RS604 Pins	Encoder Signals	RS2/RS608 Pins	Encoder Signals	RS3/RS613 Pins	Encoder Signals
1 – 2	A1 (X axis)	1 – 2	I1 (X axis)	1 – 2	B2 (Y axis)
3 – 4	A3 (Z axis)	3 – 4	I3 (Z axis)	3 – 4	B4 (W axis)
5 - 6	B3 (Z axis)	5 – 6	A4 (W axis)	5 – 6	I2 (Y axis)
7 – 8	B1 (X axis)	7 - 8	A2 (Y axis)	7 - 8	I4 (W axis)

Date: 30.03.2005

Revised: 29.09.2010

¤

Serial Interface

Subject: POSYS® 800-B/900 Series, Serial Interface

The POSYS® 800-B/900 Series motion controllers are equipped with a parallel and serial interface. Since the POSYS® 800-B is a ISA-Bus based card and the 900 is a PCI-bus card it will normally be operated using the parallel bus. The asynchronous serial connection can be used for diagnostics or stand-alone operation of the controller.

The default configuration is for parallel operation. For serial communication the only jumper (JP4., marked Mode) on the POSYS® card needs to be changed to pins 2-3.

Mode
1 – 2 Parallel
2 – 3 Serial

A special serial link cable, available upon request from POSYS Motion Control, needs to be plugged into the connector marked Serial I/F. Two types of cable are available, the standard is a RS232 cable with a 9-pin SUBD male or female connector. If the female connector is used it can connect directly to PC's serial input without a null modem.

The optional cable is for RS422/485 data transmission. The mating connector of the host must match the pinout of the supplied POSYS® connector.

RS232, Mode Setting 2-3

With this setting the motion controller powers up in the serial communication mode. At this point default parameters are active:

9600 Baud
No Parity
One Stop Bit
Point-to-Point Mode

The following commands apply to the serial communications mode:

SetDiagnosticPortMode / GetDiagnosticPortMode
SetSerialPort / GetSerialPort

Both commands are explained in detail in the manual. This will just outline the purpose of every command. Get commands return programmed values.

SetDiagnosticPortMode:

When set to LIMITED only limited access is possible like for all Get.. instructions and SetBufferReadIndex

When set to FULL, all instructions may be executed.

SetSerialPort:

Five parameters are passed to this function, they are:

Transmission rate: 1200, 2400, 9600, 19200, 57600, 115200, 250000, or 416667
Parity: NONE, ODD or EVEN
Stop Bits: 1 or 2
Protocol: Point-to-Point, Multi-drop (address bit), Multi-drop (idle line mode)
Multi-drop Address: Address 0, Address 1, …, Address 31

Checksums are needed for serial communication. A complete and detailed description of the following important subjects can be found in the NAVIGATORTM User's Guide:

Command Format
Checksums
Transmission Protocols
Point-to-Point Mode
Multi-drop Protocols
Idle-line Mode

Address Bit Mode

It is important to adhere to these protocols for an error free communication. The power-on default serial port parameter can be changed. In most cases this would be in regard to transmission rate and number of stop bits.

Important Note

For stand-alone operation without a computer and it's associated power supply it is possible to supply the POSYS® 800-B/900 series motion controller with external power using either the 2x 50-position cable or the 100-position high density connector. The best way is to use it in conjunction with IO700/800 interconnect module.

The POSYS® 800-B motion controller card must be connected to the IO700/800 using the 2x 50-position cable.

Screw Terminals 7, 57, 25 and 75: 5VDC Input, any of these inputs can be used
Screw Terminals 12, 62: GND, any of these can be used for GND connection

As an alternative the card could be used with a PC/104-bus connector and only 5 VDC and GND connected to the ISA-bus connector.

The POSYS® 900 motion control card must be connected to the IO700/800 using the shielded cable.

Screw Terminals 7, 57, 25 and 75: 5VDC Input, any of these inputs can be used
Screw Terminals 12, 62: GND, any of these can be used for GND connection

As an alternative the card could be used with a PCI-bus connector and only 5 VDC and GND connected to the PCI-bus connector.

Date: 24.03.2003

Revision:

E-Mail: support@halbeck.com

POSYS® is a registered trademark of POSYS Motion Control GmbH & Co.KG

¤

Status Read Operation

Subject: POSYS® 700/800/800B/900/1800/1900

All above POSYS motion controllers provide a Status Read function. This function returns a status word. The execution of this function can take place anytime and does not require checking of any busy flags which would cause delays. The Status Read function (POSYS function GetProcessorStatus()) returns a word containing the following information:

Bit Number	Description
0 - 12	Unused, set to 0
13	Holds value of HostIOError signal, this bit will be high if a HostIOError occurred. This is reset after reading the actual HostIOError.
14	This bit holds the value of HostInterrupt signal. A high indicates an active host interru output.
15	This bit holds the value of the HostReady signal. A high indicates *ready*.

Bit 14 requires some special attention. It can be used in a polling setup to determine if an interrupt condition, as set with SetInterruptMask(), has occurred. It can also be used to check if an interrupt was caused by the POSYS motion controller or another device.

No specific additional attention is required for all POSYS® 700/800B/900 motion controllers using the Navigator version 1.x (for example 21400018), where 18 represents version 1.8, and 20, 21, 22, 23 those with version numbers 2.0 to 2.3.

All POSYS 700/800B/900 motion controllers using the Navigator with versions 2.4 and up require an additional read or write command to assure an update of the Host Interrupt bit. The following is an example of an interrupt handler implementing such a read:

```
void DLLCALLCONV POSYS_IntHandlerRoutine(POSYS_HANDLE hPosys, POSYS_INT_RESULT *intResult)
{
   ignore_result=GetEventStatus(hPosys,0);                  //inserted command for status word update
   POSYS_Status=GetProcessorStatus(hPosys);
   if ((POSYS_Status & 0x4000)==0x00)                       //not a POSYS interrupt
      return;

   printf("Status Word: %x\n",POSYS_Status);

   if ((POSYS_Status & 0x2000)==0x2000)
     printf("HOST IO Error: %x\n",GetHostIOError(hPosys));
     printf ("Got interrupt number %d\n", intResult->dwC ounter);

   ResetEventStatus(hPosys,0,0);

   ClearInterrupt(hPosys);

   ignore_result=GetEventStatus(hPosys,0);                  //inserted command for status word update
   POSYS_Status=GetProcessorStatus(hPosys);                 //Bit 14 should now be reset

   printf("Status Word: %x\n",POSYS_Status);
}
```

POSY_STATUS contains the value of the status read. The POSYS generated the interrupt if Bit 14 is set, if not then the interrupt source is a different one.

For the POSYS 1800 and POSYS 1900 motion controllers the same requirements apply as shown in the example of the POSYS_IntHandlerRoutine above. A command must be sent to the POSYS prior to calling GetProcessorStatus() if the present state of bit 14 (interrupt) is to be returned.

It is also important to note that prior to calling C learinterrupt() all bits set with SetInterruptMask() must be cleared by calling ResetEventStatus.

The relevant interrupt commands are:

SetInterruptMask(axis)
GetInterruptAxis(axis)

```
ResetEventStatus(axis)
ClearInterrupt()
GetProcessorStatus()

Ignore_result = GetEventStatus(axis)      // dummy routine to assure that
                                          // Bit 14 in the status word returned by
                                          // (GetProcessorStatus())is updated
GetProcessorStatus()                      // returns the updated status
```

POSYS® 900 and 1900 only

The PCI motion controllers provide an additional function to check the hardware for an active interrupt line. The command is

```
INT P9050_ReadReg(hPlx, P9050_INTC SR)      or depending on the software used
INT POSYS_ReadReg(hPlx, POSYS_INTC SR)      for the POSYS 900 series
or
INT POSYS_ReadReg(hPlx, POSYS_INTC SR)      depending on the software used
```

These functions address the PLX interface directly and return the status of the Interrupt C ontrol Service Register of the PLX. Bit 2 indicates if the interrupt line 1 is active (1=active, 0=not active). Interrupt line 1 is connected to the interrupt output of the motion control processor.

Date: 26.09.2006
Revised: 05.10.2006, 09.10.2006

¤

Synchronization of Multiple Cards (POSYS® 8xx-B/9xx w/ Sync-Feature and 18xx/19xx)

Subject: POSYS® 800-B/9xx/18xx/19xx Series, Synchronization of multiple cards

The POSYS® motion controllers 8xx-B/18xx and 9xx/19xx are available with a synchronization feature. Synchronization must be specified with the order (8xx-B and 9xx) since a different motion control processor is required. When this feature is installed and activated by software it will allow a synchronized update between multiple cards to within 10 microseconds.

This feature is specified by increasing the second digit in the part number by 1. For example a POSYS® 904 becomes a POSYS® 914 signaling the synch feature. The synch feature includes additional hardware (connectors and cable) and additional firmware with two more commands. Up to 128 axes on the adequate amount of cards (1, 2 and 4 axes cards can be combined) can be synchronized, one card would be the master and the other ones the slaves. The POSYS® 18xx/19xx can combine up to 128 cards/512 axes to be synchronized. Synchronization between 8xx-B/9xx and 18xx/19xx is not possible due to different cycle times of the motion processors

The two additional commands are:

SetSynchronizationMode (mode)

mode = 0 (Disabled)
= 1 (Master)
= 2 (Slave)

mode = GetSynchronizationMode

mode will return the synchronization state of the selected card.

To allow proper use of this feature the following steps should be observed:

- Default configuration of all installed cards after power on or reset is DISABLED
- Assure that all cards are connected using the supplied 2-lead cables. The connectors can only be connected one way. The two synch connectors on each card are in parallel, so it does not matter which one is used for input or output.
- The sample times of cards used for synchronization must be the same. So assuming one 4 axes and one 2 axes card are installed. The default sampling times would be 400 and 200 microseconds. In this case the sampling time of the 2 axes card has to be adjusted to the same value as the 4 axes card. The command to use is SetSampleTime(time), where time would be in microseconds, in this case 400.
- Only one card can be the master in a chain. If 3 cards are installed and connected via the synch cable, one card will be the master and the other two will be slaves.
- The slaves must be set before the master. This ensures that the slave or slaves cycle if more than one starts at precisely the moment that the master assumes its state as master.

Example: For example assume two cards are installed, card1 will be the master, card2 will be the slave. The code is for Visual Basic.

```
Dim disabled as integer
Dim slave as integer
Dim master as integer
Dim time1 as long
Dim time2 as long

disabled = 0
master = 1
slave = 2

Call CardHandle(1) //this is the handle to card1, all data will go to this card
Call SetSynchronizationMode(disabled) //make sure the card is not in synchronization mode
time1 = GetSampleTime
Call CardHandle(2) //handle to card2, all data will go to this card
time2 = GetSampleTime
if time2 <> time1 then
Call SetToEqualTime //this calls a subroutine for adjusting the slave s sample
End If //time to the same value as the master s. This routine is not //shown as part of this example
```

```
Call CardHandle(2) //switch to card2
Call SetSynchronizationMode(slave) //set card2 to slave
Call CardHandle(1) //switch to card1
Call SetSynchronizationMode(master) //set card1 to master
```

The two cards are now synchronized.

The following sequence of commands serves as an example for starting a synchronized move across two cards.

```
Call CardHandle(1) //switch to card1
Call SetPosition(axis1, 50000) //buffer a new target position for axis1 of card1
Call CardHandle(2) //switch to card2
Call SetPosition(axis1, 25000) //buffer a new target position for axis1 of card2
Call CardHandle(1) //switch to card1
time1 = GetTime //use the master's node time to set breakpoints for both //boards that will start motion 20 servo cycles from now
Call SetBreakpointValue(axis1, Breakpoint1, time + 20)
Call SetBreakpoint(axis1, Breakpoint1, axis1, update, breakpointtime)
Call CardHandle(2) //switch to card2
Call SetBreakpointValue(axis1, Breakpoint1, time + 20)
Call SetBreakpoint(axis1, Breakpoint1, axis1, update, breakpointtime)
```

When the breakpoint triggers 20 servo cycles from now, both axes will start to move.

The master node outputs a synchronization pulse once every 50 microseconds. Applications can use this pulse to synchronize external equipment to the internal cycle of the motion processor.

Date: 02.05.2003

Revision:
E-mail: Europe: support@halbeck.com

POSYS® is a registered trade mark of POSYS Motion Control GmbH & Co.KG

¤

Velocity, Acceleration and Jerk Calculation

Subject: POSYS® Series, Programming Velocity and Acceleration

DLL Functions

All of the POSYS® motion controllers work with a servo cycle (for servo motors) or a cycle time (for stepping motors). These cycle times determine the update rate of the controller. This time is typically 100 µseconds per enabled axis (exact value is 102.4 µseconds). Some versions have a cycle time of 150 µseconds. Please refer to the appropriate data sheet.

Programming speed and acceleration as well as jerk must take this into consideration. Let us assume the following:

Motion Controller Type POSYS® 852 (2 axes stepping motor)

Fastest cycle time 2 x 102.4 µseconds = 204.8 µseconds for 2 enabled axes

The rate is 1/204.8 = 4882.81 updates/second

To program a velocity of 1000 steps/second we need to convert this to steps/update rate.

Steps/sec = (1000/4882.81) * 65536 = 13423.73, rounded this gives 13424

This would be the value sent to the card using the DLL function

SetVelocity(axis, 13424)

The easiest way to proceed would be to set up a constant like

vel_factor = 13.4217728 and write the following

SetVelocity(axis, round(1000 * vel_factor))

If the cycle time is changed the value of the constant has to be changed too.

Programming acceleration/deceleration follows a similar scheme. Assuming we want to accelerate linearly to 1000 steps/sec in 20 milliseconds. This requires an acceleration of 1000/0.020 = 50000 steps/sec². The value that needs to be loaded is calculated as follows:

Steps/sec/sec = (50000/4882.81/4882.81) * 65536 = 137.45, this is rounded to 138 and loaded using the DLL function

SetAcceleration(axis, 138)

Again we could set up a constant

accel_factor = 0.002748779 and write the following

SetAcceleration(axis, round(50000 * accel_factor))

If the cycle time is changed the value of the constant has to be changed too.

The multiplier 65536 in the above examples is needed to take care of the fractional part. For example velocity (counts/cycle) has a range of –32768 to 32767 with a resolution of 1/65536 counts/cycle.

The easiest way to set up the constants for velocity and acceleration is to use the

GetSampleTime function

It returns the servo cycle time in microseconds. Assuming a servo motor controller with 4 axes enabled would return as the default value

410 microseconds.

To obtain the correct multiplier for programming speed and acceleration per second proceed as follows:

Velocity vel_factor = (GetSampleTime/1000000) * 65536

Acceleration/Deceleration accel_factor = (GetSampleTime/1000000)² * 65536

Jerk jerk_factor = (GetSampleTime/1000000)³ * 65536

To program a speed of 10355 counts/sec. send the following command to the controller:

SetVelocity(axis, round(vel_factor * 10355)

The actual value loaded would be SetVelocity(axis, 278236)

This way the speed in counts/sec would be independent of the servo cycle time assuming that the factors would always be recalculated after a change in servo cycle time.

Example: Four axes controller with a servo cycle time of 410 µsec (exact 409,6 µsec)

Calculate factors to program in units/second.

vel_factor = (409.6/1000000) * 216 = 0.268435456

accel_factor = (409.6/1000000)² * 216 = 0.010995116

jerk_factor = (409.6/1000000)³ * 232 = 0.295147905

Now program: SetVelocity(axis, round(speed * vel_factor))

SetAcceleration(axis, round(acceleration * accel_factor))

SetJerk(axis, round(jerk * jerk_factor))

Date: 09.04.2002
Revised: 09.07.2002
Revised: 05.11.2003

¤

MotionScript®

Overview; MotionScript®

Overview; MotionScript®

MotionScript® for POSYS® is a Basic-like script language. Practically all of the GW-Basic/Quick Basic functionality with certain extensions is available. All mathematical functions can be used and user functions can be written and called from the program. One of the main features is that all of the POSYSxxx.DLL functions can be called from the script.

What is the purpose of MotionScript®?

MotionScript® allows the user to test the POSYS® motion controller hardware and write some code using POSYSxxx.DLL functions literally in a matter of minutes. Fundamental knowledge of program flow would be helpful but there is no need to be an expert C/C++ or Delphi programmer.

Portability

The program functions from the POSYSxxx.DLL are used the same way in the MotionScript® as they would be used in C/C++, Delphi or Visual Basic.

Simple Setup

PCI Bus

After the drivers for the POSYS® 9xx/19xx PCI motion controller are installed it is possible to start immediately to communicate with the motion controller. The required code is minimal:

```
if DetectPosysDevices() = 0 then                Only valid for POSYS® 900/1900 (PC I-bus)
  Print "No POSYS® controllers found!"
  end                                           REM program terminates
endif
hPosys# = OpenDevice(0)                         REM opens driver instance 1 for controller 1
                                                REM hPosys is the handle to this controller
Print "Version: "; Hex$(GetVersion(hPosys#))    REM returns as hex number the motion processor version
CloseDevice(hPosys#)                            REM always close the driver when program terminates
                                                REM program terminated

End
```

ISA/PC-104 Bus

For the POSYS® 800/800B/1800 and POSYS® 700 motion controllers the installed boards and their addresses are set using the DIP switches, must first be registered. To do this use the supplied utility program called POSYSDriverXSetup. Select the card number to register, pick the address set on this board and click on Store Data. This needs to be done only once since the data is written to the Window's registry. It will be automatically loaded after a restart or boot of the computer. It is now possible to communicate with the motion controllers, provided there are no address conflicts.

```
if RegisteredPosysDevices() = 0 then            Only valid for POSYS® 800/800B (ISA/PC -104 bus)
  Print "No POSYS® controllers found!"
  end                                           REM program terminates
endif                                           REM open the driver
if OpenDevice() = 0 then                        REM if return equals 0
  Print "Driver could not be opened."
  Print "END"
  end
endif
# hPosys = EnableIO(1, 8)                       enable ISA bus IO for the first card and get handle; no
                                                longer needed with Windriver driv er
if hPosys# = -1 then                            If return equals -1 IO range could not be enabled

  Print "IO range could not be enabled."
  Print "END"                                   REM returns as hex number the motion processor version
  end                                           REM always close the driver when program terminates
endif                                           REM program terminated
```

```
Print "Version: "; Hex$(GetVersion(hPosys#))
CloseDevice(hPosys#)
End
```

Comments can be added with the keyword REM or the character '. They are valid until the end-of-line.

Additional Information

The description of POSYS® functions in this help file may refer to hPosys, the handle being returned by a call to OpenDevice(). This is only true for the PCI bus motion controllers. For the ISA/PC-104 bus motion controllers we like to refer to the handle as hPosys, the handle being returned by a call to InitializePOSYS(hPosys, Cardaddress). The actual call to the function does not care it remains the same.

Examples:

Other POSYSxxx.DLL and specific MotionScript® functions and calculations can follow.

That is all there is to it.

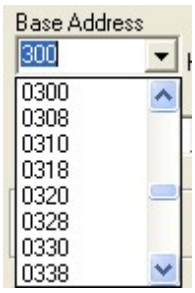
¤

POSYSDriverXSetup

POSYSDriverXSetup Program

The utility program POSYSDriverXSetup will enter base address and address range for up to 16 motion control cards into the Window's registry. The utility program can be used to enter or modify setup information or remove it.

The program will detect the installed operating system (Windows 98/Me or 2000/XP) and display this. Pick the address (Hex value) from the combo box Base Address that matches the address set on the POSYS® motion control card already installed or to be installed in your computer. Select the card number from the combo box to be associated with your motion control card. Interrupt handling is presently not supported. Address conflicts are not detected.



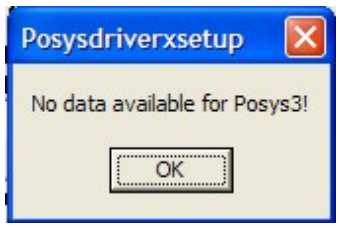
Pick your card's address from this box.

Pick the associated motion control card from this box.

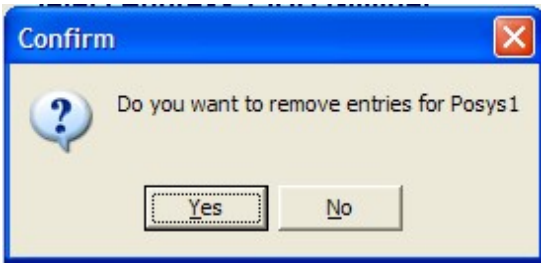
If interrupt handling is enabled in your driver you can pick an interrupt number from this box.

If this value is defined and set to Yes (1) then the driver will ignore resource conflicts when the device is created. Normally, device creation is aborted on any resource conflict.

Clicking on Store Data will write the entries to the registry, clicking on Retrieve Data will display the saved data. If no data is available the following message will be displayed:



If you click on Remove for a card that had been registered you will be asked to confirm the removal of the data.



Clicking on the Exit button will close the program.

The data written to the registry is needed by the driver to enable the IO range to communicate with the motion control board.

¤

Program Use

Program Use

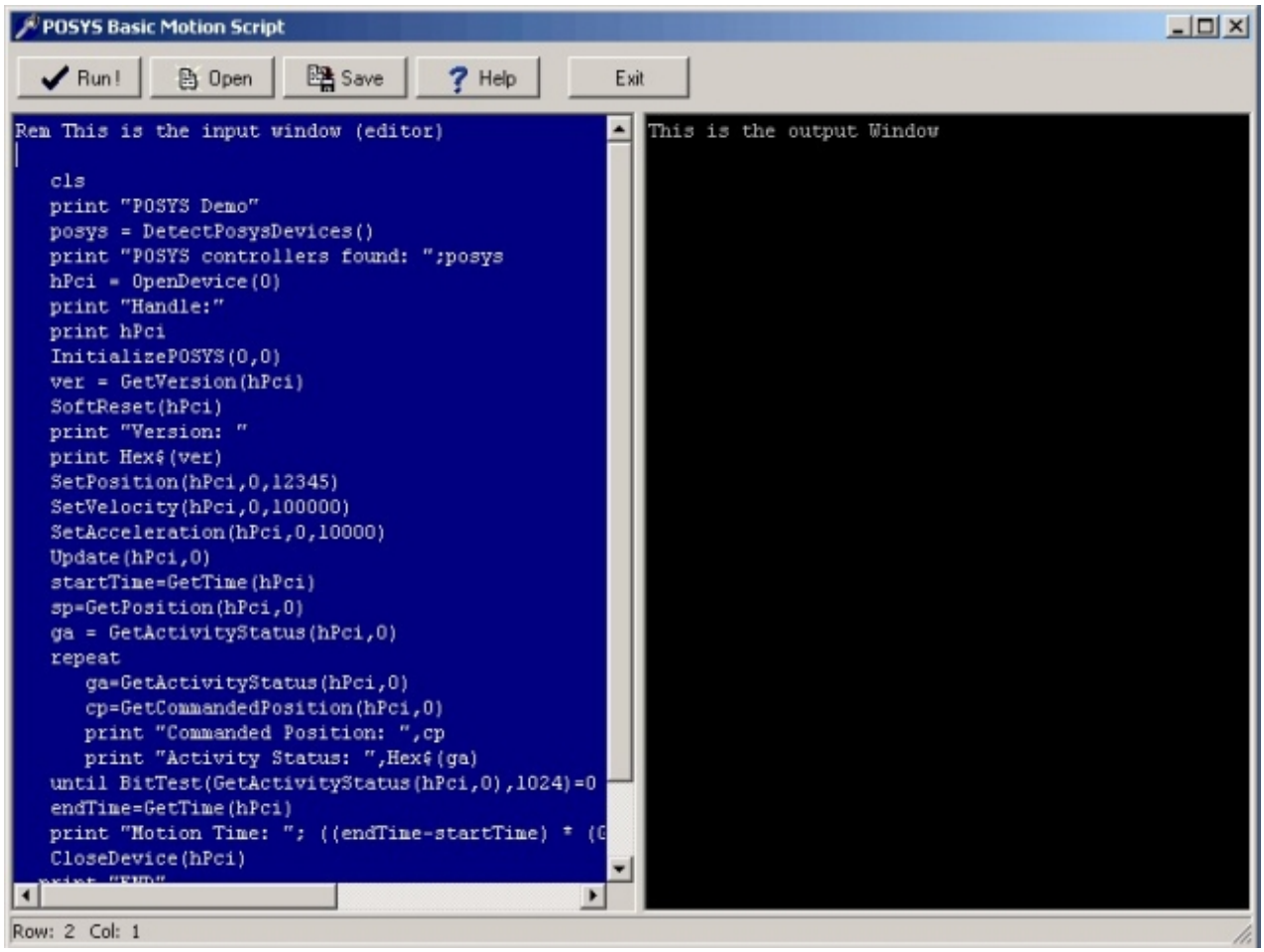
After a successful installation the program can be started immediately unless you were instructed by the operating system to shut down and restart the computer first. MotionScript® for the POSYS® is available as a stand-alone program called MotionScript® or as a menu option of the POSYSxxxVx program.

The program displays two windows, an input window which is the program editor and an output window to which all program outputs with Print statements will be directed.

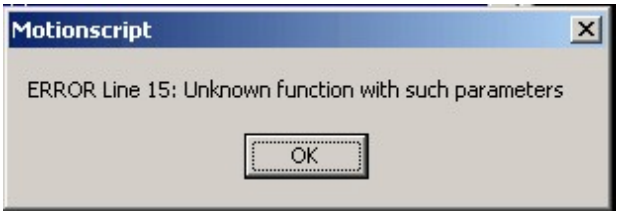
Program instructions cannot extend over a several lines, they must all fit into a single line. The length of a line is not limited. It is possible to write several instructions on a single line. Instructions must be separated by a colon (:). The following example shows how:

hPosys# = OpenDevice(0)	Rem opens driver and returns the handle in hPosys
# hPosys = EnableIO(1, 8)	This call is only needed for the ISA bus , it returns the handle in hPosys; no longer needed with Windriver driver
axis = 0	Rem defines the addressed axis (0 - 3)
readPos = GetActualPosition(hPosys#,axis) : Print "Actual Position: "; readPos	Rem returns the actual position of axis 0

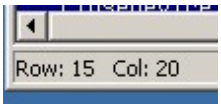
Screenshot of MotionScript®



Clicking on this button will compile the program displayed in the input window and if there are no errors execute it.



Most syntax errors are flagged and the line number with the error will be displayed.



The line the cursor is on is shown in the left hand corner of t bottom status bar. Correct any errors and click Run! again.

Clicking on this button opens a dialog and allows saving the contents (source code) of the input (editor) window.

Clicking on this button opens a dialog and loads a source file into the input (editor) window. The data will not be appended but replace existing data. The source code may be written in any text editor and then loaded for compilation and executio

Clicking on this button displays this Help file.

Clicking on this button will terminate MotionScript®. Make sure all open drivers are closed and resources are freed.

¤

Examples

Examples

In order to write code that can communicate with the POSYS® motion controller certain steps need to be observed.

1. Proper installation of the hardware.
2. Correct installation of the POSYS® drivers for Windows 98/ME or 2000/XP

It is very easy to initialize the POSYS® motion controller. After power-on or a reset the unit will be initialized with certain defaults. Please refer to the information contained under Reset.

The program code follows the information in the Programmer's Manual which details all POSYS® functions of the POSYSx00.DLL (POSYS1x00.DLL). These functions can be called from the programming language of your choice like C/C++, Delphi, VB, C++ Builder or Visual C++ or from the MotionScript® program. MotionScript® also imports POSYS® functions from the POSYSx00.DLL (POSYS1x00.DLL).

Write your code in any text editor or use the MotionScript® input (editor) window. The MotionScript® editor supports most of the standard key stroke edit functions. The code written is not case sensitive. The variable AbC is the same as abc or aBc. Any text following Rem or ' is being treated as a comment.

If the program must communicate with more than one card then open an instance of the driver for every card. A different handle will be returned for every card which will be used to address just that card. Do the following

PCI Bus

```
hPosys1# = OpenDevice(0) Rem returns handle to card 1
hPosys2# = OpenDevice(1) Rem returns handle to card 2
hPosys3# = OpenDevice(2) Rem returns handle to card 3
.. etc.
```

First steps to start driver for communication with a single POSYS® PCI card.

```
CLS                                   Rem clears the output window
found = DetectPosysDevices()          Rem searches the PCI bus for installed POSYS® motion controlle
                                      Rem if 0 is returned than no POSYS® motion controllers are
if found = 0 then                     installed
  Print "No POSYS® controllers found."
  Print "Program terminates."
  end                                 Rem terminate the program
endif                                 Rem end of if block
hPosys1# = OpenDevice(0)              Rem opens driver instance for card 1, up to 16 instances (boards)
                                      can be opened
                                      Rem no communication is possible with the card
if hPosys1# = -1 then                 Rem if OpenDevice(x) returns hPosys1 = -1 then the driver could
  Print "Driver could not be opened."  not be opened
  Print "The program terminates."
  end                                 Rem the program must be aborted! This is a reserved key word c
endif                                 MotionScript®
                                      Rem end of if block
InitializePOSYS(0,0)                  Rem this instruction could be omitted, it is only included for
                                      backward compatibility
                                      Rem print the version number of the installed motion processor t
Print "Version: ";Hex$(GetVersion(hPosys1#))  the output window
                                      Rem Hex$ is a MotionScript® function converting a decimal to
                                      hexadecimal presentation
```

All POSYS900.DLL functions can now be added to the program and combined with other MotionScript® commands. It is also possible to write MotionScript® functions and call them as sub routines

Example of a user function

```
Function SetMoveParameters(hPosys#, axis,   Rem no local parameters are required
vel, accel, pos)
SetVelocity(hPosys#, axis, vel)
SetAcceleration(hPosys#, axis, accel)
SetPosition(hPosys#, axis, pos)
EndFunc
```

Assuming in your program you want to set move parameters for axis = 0 and axis = 1

```
SetMoveParameters(hPosys#, 0, 100000,      Rem for axis = 0
20000,12345)
SetMoveParameters(hPosys#, 1, 200000,      Rem for axis = 1
40000,67890)
```

hPosys is the handle to the card being addressed.

User functions can also contain local variables. They must be declared in the body of the function.

```
FUNCTION intpower(a,b) LOCAL t,r          Rem t and r exist only in this function
r = 1                                     Rem return defines the returned result of the function
for t = 1 to b
r = r * a
next
return r
ENDFUNC
```

Loop Instructions

Loop instructions can be used to react to status information or waiting for an event to take place. For example the next move should not start until the one being executed has finished.

```
Repeat
Until                                     Rem Bit 10 is cleared when In-position or alternatively you may
                                          use a While loop
While                                     Rem Bit 10 is set when motor in motion
EndWhile
```

ISA/PC-104 Bus

Please note the differences to the PCI. For the ISA bus you call OpenDevice() only once. It does NOT return a handle. If the call is successful it returns 1 otherwise 0. You must enable the IO range for each card installed and get a handle to those cards. To do this call hPosys[No] = EnableIO(cardNo, portCount). portCount is always 8, cardNo ranges from 1 to 16. hPosys[No] will hold the handles for cards 1 to 16. If hPosys returns -1 the call was not successful. For example if you want to initialize three POSYS® 800 boards you must first call POSYSDriverXSetup, enter the addresses, store the data and go to the MotionScript® program. Enter the following:

```
OpenDevice()
  hPosys1# = EnableIO(1, 8)
  hPosys2# = EnableIO(2, 8)
  hPosys3# = EnableIO(3, 8)
```

No error checking was done in this example. You can now communicate with the three boards using the three handles hPosys1, hPosys2 and hPosys3.

```
Print "Version Card 1: ";Hex$(GetVersion(hPosys1#))
Print "Version Card 2: ";Hex$(GetVersion(hPosys2#))
Print "Version Card 3: ";Hex$(GetVersion(hPosys3#))
```

These 3 commands will write the version numbers of the three motion control boards to the MotionScript® output window. Any POSYS® function or procedure can now be called. The first parameter passed is always the handle defining which card is addressed.

¤

Variables

Supported Variables

Types

MotionScript® supports three types: numbers, pointers, and strings.

In Basic there is no need to define variables, they are automatically defined by the compiler.

Basic recognizes a variable as a string if its name ends with a dollar ($) and a pointer if its name ends with a # character. See Identifiers

Numbers

A numeric variable contains a number. The number is internally represented by a floating point value with double precision (64 bit, 15 significant digits).

Numeric variables are used with operators to perform numeric expressions and are used as counters in loop instructions.

Pointers

A pointer variable contains a 32 bit pointer.

Strings

A string variable contains text. This text may be a single row or a multi-line text. The maximum theoretical size of a string variable is 2 Gbyte.

Example:

a$ = "Hello" + " World" + "!"

The value of the variable "a$" is now "Hello World!"

String variables are used with operators to perform string expressions and are used with some functions to Open and Save data from text files.

A single row of a multi-line string can be read using square brackets, example:

if a$ contains:

" This is a
text placed on
three rows "

b$ = a$[2] ; Now b$ contains: "text placed on"

Arrays

MotionScript® does not support arrays. It is still possible though to store and retrieve array data by using three MotionScript® specific functions to implement Lists, Stacks, Queues, VarArray, Files and standard Arrays as well.

v# = alloc#(elements)
elements = number of elements of type double, v# is the pointer to the allocated memory
v# = alloc#(10) reserves memory for 10 elements of the type double and returns the pointer to this array in v#

To store data into this memory use
set(v#, offset, value) will store value at offset in memory pointed to by v#

To retrieve data from memory use
data = get(v#, offset) will return the data from offset in memory pointed to by v#, data is of the type double

¤

Reserved Words

Reserved Words

This is a list of reserved MotionScript® key words.

They have the same meaning as in traditional Basic.

if, or, to, and, end, mod, cls, rem
for, not, else, then, goto, step, break, endif
local, while, gosub, until, print, input, repeat
return, endfunc, endwhile, continue, function

Comments

Comments can be added with the keyword REM or the character '
They are valid until the end-of-line. Examples:

```
REM this is a comment
' and this is too
' but shorter
```

¤

MotionScript® Utility Functions

MotionScript® Utility Functions

The following functions serve as an extension of the Basic programming language which serves as the basics for this script language. They are required to support the motion control functions.

result = BitTest(TestValue, BitMask)

This function is useful for testing the condition of a Repeat or While loop. The following example shows its use:

hPosys# is the handle to the POSYS® motion controller returned by OpenDevice(n)

```
axis = 0
Repeat
Print "Actual Position: "; GetActualPosition(hPosys#, axis)
Print "Signal Status : "; GetSignalStatus(hPosys#, axis)
Until BitTest(GetActivityStatus(hPosys#, axis), 1024)) = 0
```

Bit 10 (=1024 , 0x400) of the Activity Status register is tested, if it is set the motor is in motion (=1024), if the motor is at rest it is cleared.

As the above example demonstrates, the TestValue could be a POSYS900.DLL function returning status information.

Two additional functions are used to set and reset selected bits.

BitSet(testVar, bitToSet) will set to 1 bitToSet in testVar
BitReset(testVar, bitToReset) will set to 0 bitToReset in testVar

There are 3 additional functions which kind of duplicate the functionality of BitTest but are limited to the Activity Status, Reset Event and Signal Status registers.

result = CompareActivity(hPosys#,axisID,mask) //checks the Activity Status register

Example:

hPosys# is the handle to the POSYS® motion controller

```
axisID = 0
mask = 1024 (bit 10)
Repeat
Until CompareActivity(hPosys#,axisID,mask) = 0 Print Axis "; axisID, "in position."
result = CompareEvent(hPosys#,axisID,mask) //checks the Event Status register
result = CompareSignal(hPosys#,axisID,mask) //checks the Signal Status register
```

¤

Other MotionScript® Functions

The following functions are integrated in the MotionScript® language.

Numeric functions

ABS(n)	Absolute value
LN(n)	Natural logarithm, base e
LOG10(n)	Logarithm, base 10
LOG2(n)	Logarithm, base 2
EXP(n)	Exponential
RND()	Random number in the range 0..1
RND(n)	Random number in the range 0..n
INT(n)	Integer part of n
FIX(n)	Integer part rounded toward zero
ROUND(n)	Rounding of n
FRAC(n)	Fractional part of n
SQR(n)	Square root
SGN(n)	Sign of n, +1 0 -1
SIN(n)	Sine of number, radians
COS(n)	Cosine
TAN(n)	Tangent
ASIN(n)	Inverse sine
ACOS(n)	Inverse cosine
ATAN(n)	Inverse tangent
ATAN2(n,m)	Inverse tangent of n/m with right quadrant
SINH(n)	Hyperbolic sine
COSH(n)	Hyperbolic cosine
TANH(n)	Hyperbolic tangent
ASINH(n)	Inverse hyperbolic sine
ACOSH(n)	Inverse hyperbolic cosine
ATANH(n)	Inverse hyperbolic tangent
DEGTORAD(n)	Degrees to radians conversion
RADTODEG(n)	Radians to degrees conversion
ASC($)	Ascii code of char
LEN($)	Length of string
VAL($)	String to number conversion

String functions

LCASE$($)	Lowercase conversion
UCASE$($)	Uppercase conversion
LTRIM$($)	Left blanks deletion
RTRIM$($)	Right blanks deletion
CHR$(n)	Ascii to char conversion
STR$(n)	Number to string conversion
HEX$(n)	Number to hexadecimal
MID$($,n,[n])	Substring extraction
LTAB$($)	Left insertion of a Tab
RTAB$($)	Right insertion of a Tab
LEFT$($,n)	Left part of a string
RIGHT$($,n)	Right part of a string

String functions, multi-rows

COUNT($)	Count rows in a multi-row string
OPENTEXT$($)	Open a text file into a string
SAVETEXT$($)	Save the string as a text file
COPYTEXT$($)	Copy the string to the clipboard
PASTETEXT$($)	Paste text from clipboard into the string
FINDFILE$($,$)	Get a list of files in the directory (dir,extension)
FINDDIR$($,$)	Get a list of directories

Miscellaneous functions

NUMBER(#)	Pointer to number conversion

POINTER(n)	Number to pointer conversion
PAUSE(n)	Suspend execution for n seconds
MESSAGE($)	Show a message in a dialog window.

¤

Loop Instructions

Loop Instructions

MotionScript® supports three kinds of loop instructions.

For, While and Repeat.

The easiest way to understand a Loop instruction is to look at an example. The following program prints the numbers 1 to 10, "a" is the control variable:

```
FOR a = 1 to 10
PRINT a
NEXT a
```

The BREAK instruction:

terminates instantly a cycle, so in this case the cycle stops when "a" becomes greater than 5.

```
FOR a = 1 to 10
PRINT a
IF a>5 then BREAK
NEXT a
```

The CONTINUE instruction:

acts as a NEXT instruction; the variable "a" is incremented and a new iteration is started immediately.

```
FOR a = 1 to 10
PRINT a
IF a>5 then CONTINUE
PRINT "hello"
NEXT a
```

FOR-NEXT instruction:

The complete syntax for this command is:

```
FOR = to [step ]
...
...
NEXT [ (ignored)]
```

examples:

```
FOR angle = 1+ASIN(0.4) to 1+ASIN(0.75) step 0.1
PRINT angle
NEXT
FOR y = 1 to 200
FOR x = 1 to 320
s = s + x*y
NEXT
NEXT
```

REPEAT-UNTIL instruction:

This instruction repeats a series of instructions until the condition is True. The block begins with the REPEAT command and ends with the UNTIL command followed by the condition that is evaluated.

Example:

```
a = 1
REPEAT
PRINT a
a = a+1
UNTIL a>10
```

The Repeat instruction evaluates the condition at the end of the loop, this means that the instructions inside the loop are executed at least once. The Break and Continue instructions can be used in REPEAT-UNTIL cycles just like in the FOR-NEXT cycles.

WHILE-ENDWHILE instruction:

This instruction evaluates a condition at the beginning of the loop. If the condition is false then the cycle stops and execution continues after the EndWhile instruction.

Example:

```
a = 1
WHILE a <= 10
PRINT a
a = a+1
ENDWHILE
```

Since the While command evaluates the condition at the beginning of the loop, the instructions inside the loop may never be also executed. The Break and Continue instructions can be used in WHILE-ENDWHILE cycles just like in the FOR-NEXT cycles.

¤

Writing MotionScript® Functions

A function is defined as set of instructions that computes and returns a value.

The result can be discarded (like C and Delphi) or simply not returned, making it a procedure. Local variables can be defined to support recursive functions.

Syntax:

```
Function ( parameters ) [ Local ]
...
[Return ]
...
Endfunc
```

Examples:

```
FUNCTION sum(a,b)
return a+b
ENDFUNC

FUNCTION intpower(a,b) LOCAL t,r
r=1
for t = 1 to b
r = r * a
next
return r
ENDFUNC

FUNCTION hello$(a$)
return "Hello to you " + a$
ENDFUNC
```

MotionScript® fully supports recursive functions.

Example:

```
FUNCTION factorial(n)
if n <= 1 then return 1
return n*factorial(n-1)
ENDFUNC
```

¤

POSYS® Functions and Procedures

POSYS® Functions and Procedures

This is a list of all available POSYS® functions for the MotionScript® language. These functions are imported from the POSYS900.DLL for the POSYS® 900 series and from the POSYS1900.DLL for the POSYS® 1900 series. The listing is from the import file Imp900.pas for Delphi. MotionScript® itself does not require an import file, it is an integral part of the program.

Explanation:

Procedure is a Function without a return value.
Function is a Function with a return value.

hPosys refers to the card handle regardless of talking to a PCI bus or ISA/PC-104 bus card. The only difference is in the way of getting the handle. For the PCI bus hPosys = OpenDevice(deviceNo) returns the handles for deviceNo 0-15. For the ISA/PC-104 bus hPosys# = EnableIO(deviceNo, 8) returns the handles for deviceNo 1-16. You may use any name of your choice for the variable hPosys. In C/C++ and Delphi they need to be declared as integers. In MotionScript® no prior declaration to first use is required.

Parameters:

type
Thandle : integer

Delphi	C/C++	Visual Basic
integer = 32 bits	INT	LONG
word = 16 bits	SHORT	INTEGER
byte = 8 bits	CHAR	BYTE

Example for the different calling conventions:

hPosys is in all of the examples the handle to the POSYS® motion control card returned by OpenDevice (device)

```
hPosys = OpenDevice(device1)          //for example device1 = 0 for the first instance
```

For the ISA/PC-104 bus motion controllers the handle is returned by a call to

```
hPosys = EnableIO(device1, 8)       //device range: 1 - 16, for example device1 = 1
```

In the POSYS® DLLs hPosys was chosen to declare the handle for PCI bus communication. It could also be used for the ISA bus communication but will hold an entire different value (ISA bus address) than in the PCI bus communication. To make the difference clear this manual will always refer to the ISA bus handle as hPosys

```
hPosys = EnableIO(device1, 8)
```

MotionScript® Syntax

```
axis = 1
position = 12345
returnResult = 0

'calling a Procedure without a return value:
```

```
AdjustActualPosition(hPosys#, axis, position)
'or for the example
AdjustActualPosition(hPosys#, 1, 12345)

'calling a Function with a return value:
returnResult = GetPositionError(hPosys#, axis)
'or for the example
returnResult = GetPositionError(hPosys#, 1)
```

Delphi

```
Var
axis :word = 1;
position :integer = 12345;
returnResult :integer = 0;

// calling a Procedure without a return value:
AdjustActualPosition(hPosys, axis, position);
//or for the example
AdjustActualPosition(hPosys, 1, 12345);

// calling a Function with a return value:
returnResult := GetPositionError(hPosys, axis);
//or for the example
returnResult := GetPositionError(hPosys, 1);
```

C/C++

```
SHORT axis = 1;
INT position = 12345;
INT returnResult = 0;

// calling a Procedure without a return value:
AdjustActualPosition(hPosys, axis, position);
//or for the example
AdjustActualPosition(hPosys, 1, 12345);

// calling a Function with a return value:
returnResult = GetPositionError(hPosys, axis);
//or for the example
returnResult = GetPositionError(hPosys, 1);
```

Visual Basic

```
Dim Axis as Integer
Dim position as Long
Dim returnResult as Long
axis = 1
position = 12345
returnResult = 0

' calling a Procedure without a return value:
Call AdjustActualPosition(hPosys, axis, position)
'or for the example
Call AdjustActualPosition(hPosys, 1, 12345)

' calling a Function with a return value:
returnResult = GetPositionError(hPosys, axis);
'or for the example
returnResult = GetPositionError(hPosys, 1);
```

POSYS® Functions and Procedures

MotionScript® requires for the parameters the type DOUBLE and if a value is returned it is also returned as a DOUBLE. By default all numeric MotionScript® variables are of the type DOUBLE, in fact that is the only numeric type

Card Initialization & Addressing

```
Procedure InitializePOSYS(cardAddress0, cardAddress1:word);stdcall;
Function OpenDevice(device:byte):Thandle;stdcall;
```

Function OpenDevice:word;
Function CloseDevice(hPosys :integer) :integer;stdcall;
Function DetectPosysDevices:word;stdcall;
Function RegisteredPosysDevices:word;stdcall;

Breakpoints & Interrupts

Procedure ClearInterrupt(hPosys:Thandle);stdcall;
Function GetInterruptAxis(hPosys:Thandle):word;stdcall;
Procedure SetBreakpoint(hPosys:Thandle;axis,breakpoint,sourceAxis,action,trigger:word);stdcall;
Function GetBreakpoint(hPosys:Thandle;axis,breakpoint:word):word;stdcall;
Procedure SetBreakpointValue(hPosys:Thandle;axis,breakpoint:word;value:integer);stdcall;
Function GetBreakpointValue(hPosys:Thandle;axis,breakpoint:word):integer;stdcall;
Procedure SetInterruptMask(hPosys:Thandle;axis,interruptMask:word);stdcall;
Function GetInterruptMask(hPosys:Thandle;axis:word):word;stdcall;

Commutation

Function GetPhaseCommand(hPosys:Thandle;axis:word):integer;stdcall;
Procedure InitializePhase(hPosys:Thandle;axis:word);stdcall;
Procedure SetCommutationMode(hPosys:Thandle;axis,mode:word);stdcall;
Function GetCommutationMode(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetNumberPhases(hPosys:Thandle;axis,phases:word);stdcall;
Function GetNumberPhases(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetPhaseAngle(hPosys:Thandle;axis,angle:word);stdcall;
Function GetPhaseAngle(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetPhaseCorrectionMode(hPosys:Thandle;axis,mode:word);stdcall;
Function GetPhaseCorrectionMode(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetPhaseCounts(hPosys:Thandle;axis,count:word);stdcall;
Function GetPhaseCounts(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetPhaseInitializeMode(hPosys:Thandle;axis,mode:word);stdcall;
Function GetPhaseInitializeMode(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetPhaseInitializeTime(hPosys:Thandle;axis,time:word);stdcall;
Function GetPhaseInitializeTime(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetPhaseOffset(hPosys:Thandle;axis,offset:word);stdcall;
Function GetPhaseOffset(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetPhasePrescale(hPosys:Thandle;axis,mode:word);stdcall;
Function GetPhasePrescale(hPosys:Thandle;axis:word):word;stdcall;

Digital Servo Filter

Procedure ClearPositionError(hPosys:Thandle;axis:word);stdcall;
Procedure DACOutputsOff(hPosys:Thandle);stdcall;
Procedure DACOutputsOn(hPosys:Thandle);stdcall;
Function GetAmpEnable(hPosys:Thandle;ampID:word):word;stdcall;
Function GetDerivative(hPosys:Thandle;axis:word):word;stdcall;
Function GetIntegral(hPosys:Thandle;axis:word):integer;stdcall;
Function GetPositionError(hPosys:Thandle;axis:word):integer;stdcall;
Procedure SetAmpDisable(hPosys :integer; disable :word);stdcall;
Procedure SetAmpEnable(hPosys :integer; enable :word);stdcall;
Procedure SetAutoStopMode(hPosys:Thandle;axis,mode:word);stdcall;
Function GetAutoStopMode(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetBiQuadCoefficient(hPosys:Thandle;axis,coefficientID,filterID,value:word);stdcall;
Function GetBiQuadCoefficient(hPosys:Thandle,axis:word);stdcall;
Procedure SetDerivativeTime(hPosys:Thandle;axis,time:word);stdcall;
Function GetDerivativeTime(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetIntegrationLimit(hPosys:Thandle;axis:word;limit:integer);stdcall;
Function GetIntegrationLimit(hPosys:Thandle;axis:word):integer;stdcall;
Procedure SetKaff(hPosys:Thandle;axis,Kaff:word);stdcall;
Function GetKaff(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetKd(hPosys:Thandle;axis,Kd:word);stdcall;
Function GetKd(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetKi(hPosys:Thandle;axis,Ki:word);stdcall;
Function GetKi(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetKp(hPosys:Thandle;axis,Kp:word);stdcall;
Function GetKp(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetKout(hPosys:Thandle;axis,Kout:word);stdcall;
Function GetKout(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetKvff(hPosys:Thandle;axis,Kvff:word);stdcall;
Function GetKvff(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetMotorBias(hPosys:Thandle;axis,bias:word);stdcall;
Function GetMotorBias(hPosys:Thandle;axis:word):word;stdcall;

Procedure SetMotorLimit(hPosys:Thandle;axis,limit:word);stdcall;
Function GetMotorLimit(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetPositionErrorLimit(hPosys:Thandle;axis:word;limit:integer);stdcall;
Function GetPositionErrorLimit(hPosys:Thandle;axis:word):integer;stdcall;

Piezo (Ceramic) Motor Functions

Procedure SetpzOffsetPos(hPosys:Thandle;axis:word;offset:word);stdcall;external DLL;
Function GetpzOffsetPos(hPosys:Thandle;axis:word):word;stdcall;external DLL;
Procedure SetpzOffsetNeg(hPosys:Thandle;axis:word;offset:word);stdcall;external DLL;
Function GetpzOffsetNeg(hPosys:Thandle;axis:word):word;stdcall;external DLL;
Procedure SetpzIntLimitStl(hPosys:Thandle;axis:word;limit:longword);stdcall;external DLL;
Function GetpzIntLimitStl(hPosys:Thandle;axis:word):longword;stdcall;external DLL;
Procedure SetpzWindow(hPosys:Thandle;axis:word;window:word);stdcall;external DLL;
Function GetpzWindow(hPosys:Thandle;axis:word):word;stdcall;external DLL;
Procedure SetpzKp(hPosys:Thandle;axis:word;kp:word);stdcall;external DLL;
Function GetpzKp(hPosys:Thandle;axis:word):word;stdcall;external DLL;
Procedure SetpzKi(hPosys:Thandle;axis:word;ki:word);stdcall;external DLL;
Function GetpzKi(hPosys:Thandle;axis:word):word;stdcall;external DLL;
Procedure SetpzKd(hPosys:Thandle;axis:word;kd:word);stdcall;external DLL;
Function GetpzKd(hPosys:Thandle;axis:word):word;stdcall;external DLL;
Procedure SetpzKvff(hPosys:Thandle;axis:word;kvff:word);stdcall;external DLL;
Function GetpzKvff(hPosys:Thandle;axis:word):word;stdcall;external DLL;
Procedure SetpzKaff(hPosys:Thandle;axis:word;kaff:word);stdcall;external DLL;
Function GetpzKaff(hPosys:Thandle;axis:word):word;stdcall;external DLL;
Procedure SetpzIntLim(hPosys:Thandle;axis:word;intlim:longword);stdcall;external DLL;
Function GetpzIntLim(hPosys:Thandle;axis:word):longword;stdcall;external DLL;
Procedure SetpzKdPeriod(hPosys:Thandle;axis:word;kdperiod:word);stdcall;external DLL;
Function GetpzKdPeriod(hPosys:Thandle;axis:word):word;stdcall;external DLL;
Procedure SetpzWindow2(hPosys:Thandle;axis:word;window2:word);stdcall;external DLL;
Function GetpzWindow2(hPosys:Thandle;axis:word):word;stdcall;external DLL;
Procedure SetpzKvff2(hPosys:Thandle;axis:word;kvff2:word);stdcall;external DLL;
Function GetpzKvff2(hPosys:Thandle;axis:word):word;stdcall;external DLL;

Encoder

Procedure AdjustActualPosition(hPosys:Thandle;axis:word;position:integer);stdcall;
Function GetAbsoluteSSIPosition(hPosys:Thandle;axisID:word):integer;stdcall;
Function GetActualVelocity(hPosys:Thandle;axis:word):integer;stdcall;
Function GetCaptureValue(hPosys:Thandle;axis:word):integer;stdcall;
Function GetSSIVersion(hPosys:Thandle):word;stdcall;
Procedure WriteSSIRegister(hPosys:Thandle;axisID:word; resolution, frequency:word);stdcall;
Procedure ReadSSIRegister(hPosys:Thandle;axisID:word;var resolution:word;var frequency:word);stdcall;
Procedure WriteSSIResolution(hPosys:Thandle;axisID:word;var resolution:word);stdcall;
Procedure ReadSSIResolution(hPosys:Thandle;axisID:word;var resolution:word);stdcall;
Procedure WriteSSIFrequency(hPosys:Thandle;axisID:word;var resolution:word);stdcall;
Procedure ReadSSIFrequency(hPosys:Thandle;axisID:word;var resolution:word);stdcall;
Procedure ResetSSI(hPosys:Thandle);stdcall;
Procedure SetActualToAbsolutePosition(hPosys:Thandle;axisID:word);stdcall;
Procedure SetActualPosition(hPosys:Thandle;axis:word;pos:integer);stdcall;
Function GetActualPosition(hPosys:Thandle;axis:word):integer;stdcall;
Procedure SetActualPositionUnits(hPosys:Thandle;axis,mode:word);stdcall;
Function GetActualPositionUnits(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetAuxiliaryEncoderSource(hPosys:Thandle;axis,mode,auxiliaryAxis:word);stdcall;
Function GetAuxiliaryEncoderSource(hPosys:Thandle;axis:word);stdcall;
Procedure SetCaptureSource(hPosys:Thandle;axis,source:word);stdcall;
Function GetCaptureSource(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetEncoderModulus(hPosys:Thandle;axis,modulus:word);stdcall;
Function GetEncoderModulus(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetEncoderSource(hPosys:Thandle;axis,source:word);stdcall;
Function GetEncoderSource(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetEncoderToStepRatio(hPosys:Thandle;axis,encodercounts,steps:word);stdcall;
Function GetEncoderToStepRatio(hPosys:Thandle;axis:word):integer;stdcall;

On-Board RAM

Procedure EnableDPRam(hPosys:P1800_Handle);stdcall; //only valid for POSYS® 1800 series equipped with Dual Port RAM
Procedure DisableDPRam(hPosys:P1800_Handle);stdcall; //only valid for POSYS® 1800 series equipped with Dual Port RAM
Function ReadBuffer(hPosys:Thandle;bufferID:word):integer;stdcall;
Procedure SetBufferFunction(hPosys:Thandle;axis:word;funct:integer;bufferID:word);stdcall;
Function GetBufferFunction(hPosys:Thandle;axis:word;funct:word):word;stdcall;
Procedure SetBufferLength(hPosys:Thandle;bufferID:word;length:integer);stdcall;

Function GetBufferLength(hPosys:Thandle;bufferID:word):integer;stdcall;
Procedure SetBufferReadIndex(hPosys:Thandle;bufferID:word;index:integer);stdcall;
Function GetBufferReadIndex(hPosys:Thandle;bufferID:word):integer;stdcall;
Procedure SetBufferStart(hPosys:Thandle;bufferID:word;address:integer);stdcall;
Function GetBufferStart(hPosys:Thandle;bufferID:word):integer;stdcall;
Procedure SetBufferWriteIndex(hPosys:Thandle;bufferID:word;index:integer);stdcall;
Function GetBufferWriteIndex(hPosys:Thandle;bufferID:word):integer;stdcall;
Procedure WriteBuffer(hPosys:Thandle;bufferID:word;value:integer);stdcall;
Function ReadDPRamLong(hPosys:Thandle;offset:dword):integer;stdcall;
Procedure WriteDPRamLong(hPosys:Thandle;offset:dword;value:dword);stdcall;

Motor Output

Function GetCurrentMotorCommand(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetMotorCommand(hPosys:Thandle;axis,value:word);stdcall;
Function GetMotorCommand(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetMotorMode(hPosys:Thandle;axis,mode:word);stdcall;
Function GetMotorMode(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetMotorType(hPosys:Thandle;axis:word):word;stdcall;
Function GetMotorType(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetOutputMode(hPosys:Thandle;axis,mode:word);stdcall;
Function GetOutputMode(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetPWMFrequency(hPosys:Thandle;axis,frequency:word);stdcall;
Function GetPWMFrequency(hPosys:Thandle;axis:word);stdcall;
Procedure SetStepRange(hPosys:Thandle;axis,frequency:word);stdcall;
Function GetStepRange(hPosys:Thandle;axis:word):word;stdcall;

Profile Generation

Function GetCommandedAcceleration(hPosys:Thandle;axis:word):integer;stdcall;
Function GetCommandedPosition(hPosys:Thandle;axis:word):integer;stdcall;
Function GetCommandedVelocity(hPosys:Thandle;axis:word):integer;stdcall;
Procedure MultiUpdate(hPosys:Thandle;mask:word);stdcall;
Procedure SetAcceleration(hPosys:Thandle;axis:word;accel:integer);stdcall;
Function GetAcceleration(hPosys:Thandle;axis:word):integer;stdcall;
Procedure SetDeceleration(hPosys:Thandle;axis:word;deceleration:integer);stdcall;
Function GetDeceleration(hPosys:Thandle;axis:word):integer;stdcall;
Procedure SetGearMaster(hPosys:Thandle;axis,masterAxis,source:word);stdcall;
Function GetGearMaster(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetGearRatio(hPosys:Thandle;slaveAxis:word;ratio:integer);stdcall;
Function GetGearRatio(hPosys:Thandle;slaveAxis:word):integer;stdcall;
Procedure SetJerk(hPosys:Thandle;axis:word;jerk:integer);stdcall;
Function GetJerk(hPosys:Thandle;axis:word):integer;stdcall;
Procedure SetPosition(hPosys:Thandle;axis:word;position:integer);stdcall;
Function GetPosition(hPosys:Thandle;axis:word):integer;stdcall;
Procedure SetProfileMode(hPosys:Thandle;axis,mode:word);stdcall;
Function GetProfileMode(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetStartVelocity(hPosys:Thandle;axis:word;velocity:integer);stdcall;
Function GetStartVelocity(hPosys:Thandle;axis:word):integer;stdcall;
Procedure SetStop(hPosys:Thandle;axis,mode:word);stdcall;
Function GetStop(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetStopMode(hPosys:Thandle;axis,mode:word);stdcall;
Function GetStopMode(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetVelocity(hPosys:Thandle;axis:word;velocity:integer);stdcall;
Function GetVelocity(hPosys:Thandle;axis:word):integer;stdcall;
Procedure Up_date(hPosys:Thandle;axis:word);stdcall;
Procedure Update(hPosys:Thandle;axis:word);stdcall;

Servo Loop Control

Function GetTime(hPosys:Thandle):integer;stdcall;
Procedure SetAxisMode(hPosys:Thandle;axis,mode:word);stdcall;
Function GetAxisMode(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetLimitSwitchMode(hPosys:Thandle;axis,mode:word);stdcall;
Function GetLimitSwitchMode(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetMotionCompleteMode(hPosys:Thandle;axis,mode:word);stdcall;
Function GetMotionCompleteMode(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetSampleTime(hPosys:Thandle;time:word);stdcall;
Function GetSampleTime(hPosys:Thandle):word;stdcall;
Procedure SetSettleTime(hPosys:Thandle;axis,time:word);stdcall;
Function GetSettleTime(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetSettleWindow(hPosys:Thandle;axis,window:word);stdcall;

Function GetSettleWindow(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetTrackingWindow(hPosys:Thandle;axis,window:word);stdcall;
Function GetTrackingWindow(hPosys:Thandle;axis:word):word;stdcall;

Status Registers and AxisOut Indicator

Function GetActivityStatus(hPosys:Thandle;axis:word):word;stdcall;
Function GetAxis_In_Out_Home(hPosys:Thandle;statID:word):word;stdcall;
Function GetEventStatus(hPosys:Thandle;axis:word):word;stdcall;
Function GetSignalStatus(hPosys:Thandle;axis:word):word;stdcall;
Procedure ResetEventStatus(hPosys:Thandle;axis,mask:word);stdcall;
Procedure SetAxisOutSource(hPosys:Thandle;axis:word;sourceAxis,bit,reg:word);stdcall;
Function GetAxisOutSource(hPosys:Thandle;axis:word):word;stdcall;
Procedure SetSignalSense(hPosys:Thandle;axis,mask:word);stdcall;
Function GetSignalSense(hPosys:Thandle;axis:word):word;stdcall;

Traces

Function GetTraceCount(hPosys:Thandle):integer;stdcall;
Function GetTraceStatus(hPosys:Thandle):word;stdcall;
Procedure SetTraceMode(hPosys:Thandle;mode:word);stdcall;
Function GetTraceMode(hPosys:Thandle):word;stdcall;
Procedure SetTracePeriod(hPosys:Thandle;period:word);stdcall;
Function GetTracePeriod(hPosys:Thandle):word;stdcall;
Procedure SetTraceStart(hPosys:Thandle;triggerAxis,condition,triggerBit,triggerState:word);stdcall;
Function GetTraceStart(hPosys:Thandle):word;stdcall;
Procedure SetTraceStop(hPosys:Thandle;triggerAxis,condition,triggerBit,triggerState:word);stdcall;
Function GetTraceStop(hPosys:Thandle):word;stdcall;
Procedure SetTraceVariable(hPosys:Thandle;variableNumber,traceAxis,variable:word);stdcall;
Function GetTraceVariable(hPosys:Thandle;variableNumber:word):word;stdcall;

Miscellaneous (Controller Reset, Communication Mode aso)

Function EnableIO(deviceNo, portCount:word):word
Function GetChecksum(hPosys);stdcall;
Function GetHostIOError(hPosys:Thandle):word;stdcall;
Function GetProcessorStatus(hPosys:Thandle):word;stdcall;
Function Get_Version(hPosys:Thandle):integer;stdcall;
Function GetVersion(hPosys:Thandle):integer;stdcall;
Procedure HardReset(hPosys:Thandle);stdcall;
Procedure MCReset(hPosys:Thandle);stdcall;
Procedure NoOperation(hPosys:Thandle);stdcall;
Function PortInW(hPosys:Thandle;address:word):word;stdcall;
Procedure PortOutW(hPosys:Thandle;address,outword:word);stdcall;
Procedure PortOutB(hPosys:Thandle;address:word;value:byte);stdcall;
Function ReadAnalog(hPosys:Thandle;portID:word):word;stdcall;
Function ReadIO(hPosys:Thandle;ioID:word):word;stdcall;
Function ReadPLD(hPosys:Thandle;pld_adr:word):word;stdcall;
Procedure SoftReset(hPosys:Thandle);stdcall;
Procedure SetCANMode(hPosys:mask);stdcall;
Function GetCANMode(hPosys);stdcall;
Procedure SetDiagnosticPortMode(hPosys:Thandle;mode:word);stdcall;
Function GetDiagnosticPortMode(hPosys:Thandle):word;stdcall;
Procedure SetSerialPort(hPosys:Thandle;baudrate,parity,stopbits,protocol,mdaddress:word);stdcall;
Function GetSerialPort(hPosys:Thandle):word;stdcall;
Procedure SetSerialPortMode(hPosys:Thandle;baudrate,parity,stopbits,protocol,mdaddress:word);stdcall;
Function GetSerialPortMode(hPosys:Thandle):word;stdcall;
Procedure SetSynchronizationMode(hPosys:Thandle;mode:word);stdcall;
Function GetSynchronizationMode(hPosys:Thandle):word;stdcall;
Procedure SetWatchDog(hPosys:Thandle;watchdog_adr:word;watchdog_value:word);stdcall;
Procedure WriteIO(hPosys:Thandle;ioID,data:word);stdcall;

Additional MotionScript® Functions

Procedure BitReset(hPosys:Thandle;testVar,bitToReset:integer);stdcall;
Procedure BitSet(hPosys:Thandle;testVar,bitToReset:integer);stdcall;
Procedure BitTest(hPosys:Thandle;testVar,bitToReset:integer);stdcall;
Procedure CompareActivity(hPosys:Thandle;axis:word;mask:integer);stdcall;
Procedure CompareEvent(hPosys:Thandle;axis:word;mask:integer);stdcall;
Procedure CompareSignal(hPosys:Thandle;axis:word;mask:integer);stdcall;

¤

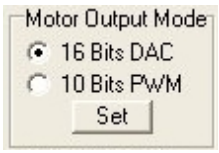
POSYS® GUI für Windows 98/ME/2000/XP

Einleitung

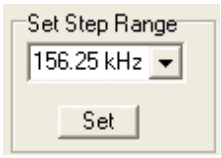
Einleitung

Das POSYS® Programm für Windows 98/ME/2000/XP bietet eine Fülle an Beispielen was die POSYS® Positioniersteuerkarten zu leisten vermögen und wie es programmtechnisch umgesetzt werden kann. Um Missverständnisse von vornherein zu vermeiden - die POSYS® GUI ist kein Anwendungsprogramm für die tägliche Arbeit, sondern soll einerseits nur aufzeigen welche Möglichkeiten die POSYS® bietet und andererseits eine erste "Sofortmassnahme" zur Inbetriebnahme der Positioniersteuerkarte bieten inklusive Ausgabemöglichkeit an Verstärker, Rückmeldung der Enkoder, Setzen/Zurücksetzen der IOs etc....

Je nachdem welche Karte und für welchen Motortyp die Software eingesetzt werden soll, kann es zu leichten Unterschieden, auf die dann aber an entsprechender Stelle hingewiesen wird, kommen. Ein Beispiel ist der Einsatz von Servo oder Schrittmotoren. Je nachdem welche Kartentype eingesetzt wird, erscheint einer der beiden "Motor Output Mode" Boxen:

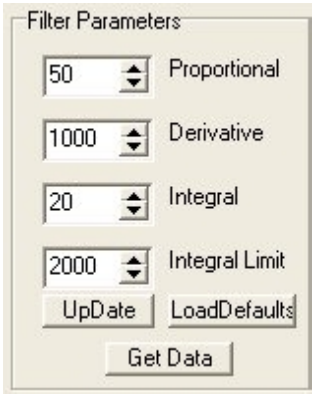


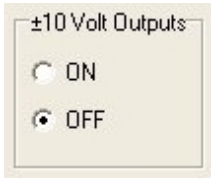
Servomotor



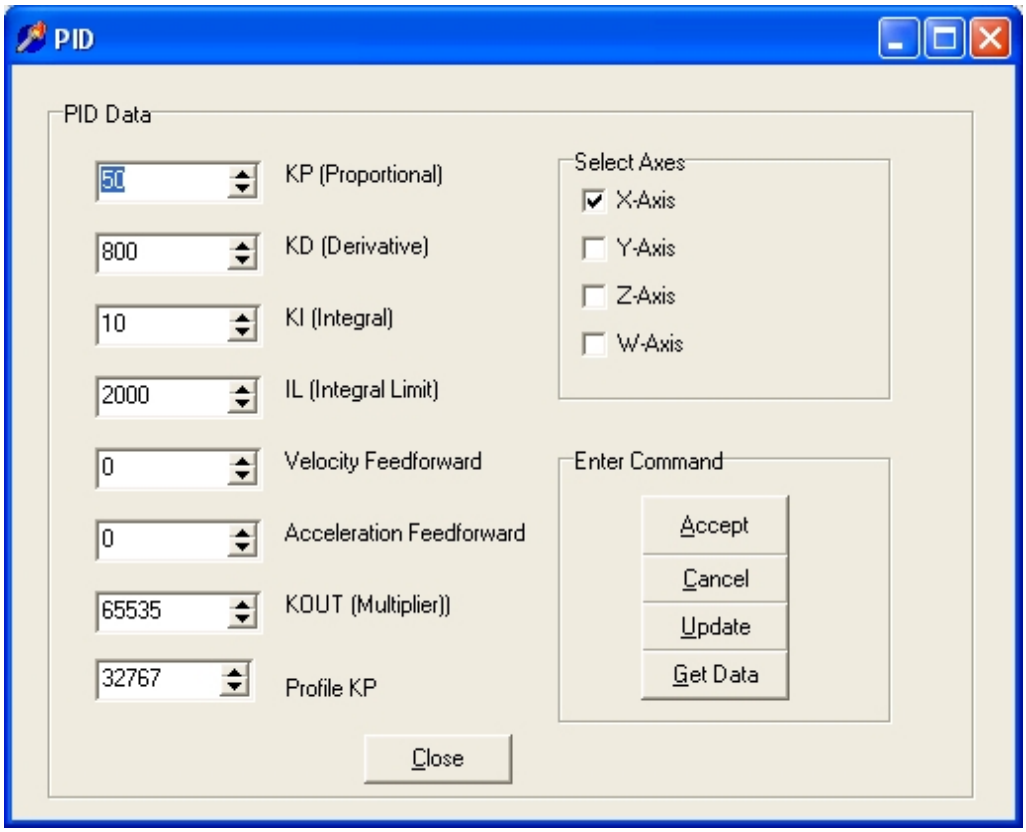
\- Schrittmotor

Weitere Änderungen die, wie in diesem Beispiel dargestellt, eintreten sind die Deaktivierung der PID-Filterparameterbox und der DACOutputs ON/OFF Funktion wenn Schrittmotorkarten erkannt werden.





Desweiteren ist über das Menü die erweiterte PID-Filterparametereinstellung nicht mehr zugänglich



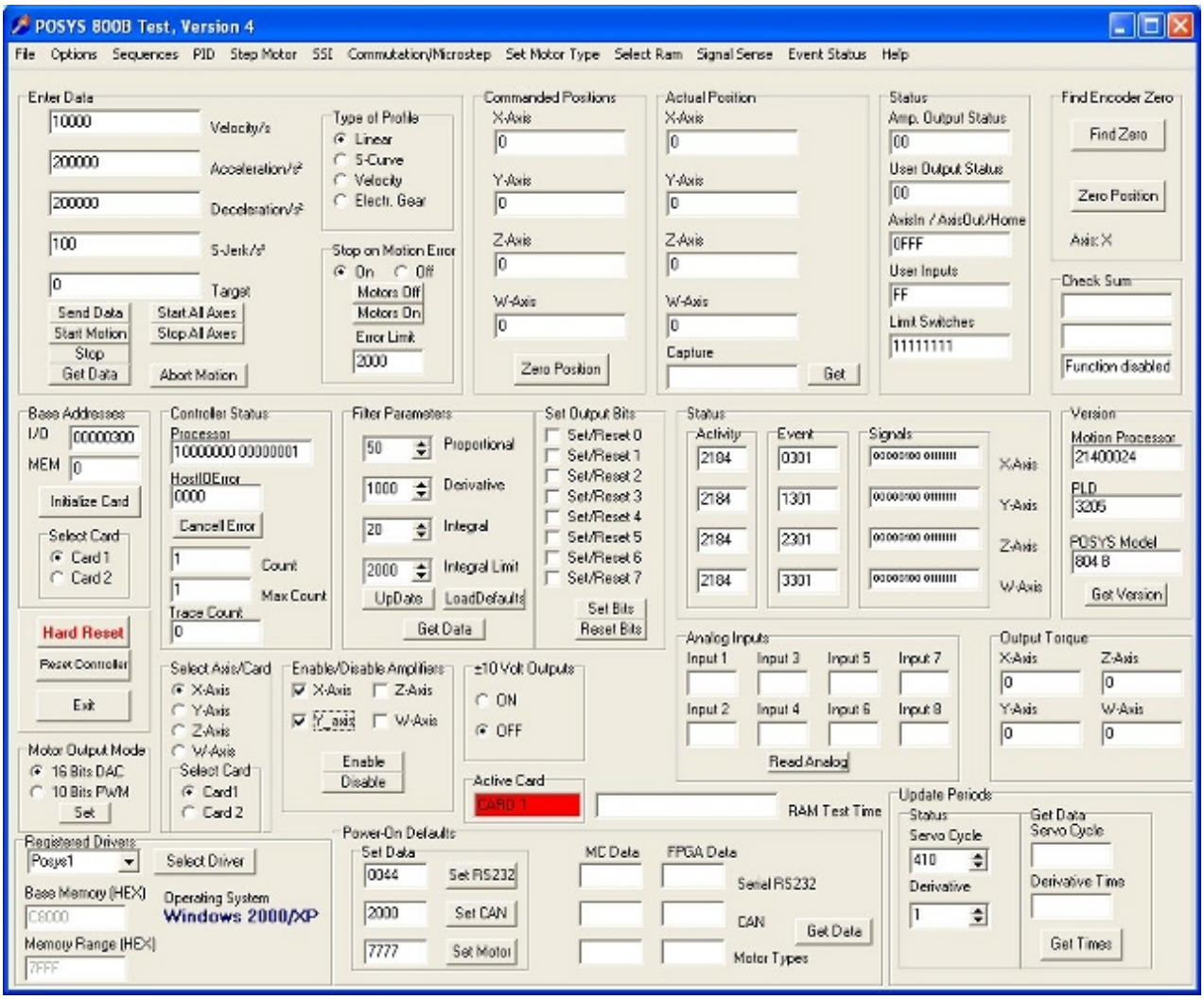
und stattdessen erscheint folgende Meldung bei dem Versuch dieses Menü auf zu rufen:

Auf den folgenden Seiten werden die einzelnen Funktionen und Möglichkeiten der POSYS® GUI näher beschrieben und erläutert.

¤

Das POSYS® GUI Hauptfenster

Das Hauptfenster des POSYS® GUI Programmes



Bevor dieses Programm gestartet und verwendet werden kann, muss mit dem Programm POSYSDriverXSetup (bei ISA-Bus basierten Karten; POSYS® 700/800/800-B/1800) der Treiber in der Registry von Windows eingetragen worden sein. Bei PCI-Bus basierten Karten (POSYS® 900/1900) wird beim Booten "neue Hardware entdeckt" gemeldet. Sie werden aufgefordert die Position der Treiber für die Karte(n) anzugeben. Wählen sie den entsprechenden Treiber aus (entweder für POSYS® 900 oder 1900) und folgen Sie den Bildschirmanweisungen von Windows für die standardmäßige Hardwareinstallation von PCI-Karten. Das mitgelieferte Programm PCI ScanforPOSYS hilft Ihnen bei der Ermittlung der zugewiesenen Adressen für die PCI-Bus basierten Karten. Ebenso ist es notwendig das im Unterverzeichnis /OCX der mitgelieferten CD befindliche Batchprogramm "registerOCX.bat", dass vorher auf die Festplatte kopiert worden sein soll, aufzurufen, um bestimmte für die Grafikausgabe notwendige Komponenten in Windows zu registrieren. Je nach Betriebssystem kann es notwendig sein 2 Windowsmeldungen manuell zu bestätigen.

Nachdem dies geschehen ist, ist es möglich obiges Programm zu starten. Um mit der POSYS® eine Kommunikation aufbauen zu können, muss nun der Treiber geladen und anschliessend die Karte initialisiert werden. Wie bereits gesagt - dies ist ein Demo-Programm. Diese Schritte können in Ihrer eigenen Applikation automatisiert ablaufen. Uns war daran gelegen dem Programmierer, den wir hier zuallererst ansprechen wollen, möglichst viele auch nebensächlich erscheinende Funktionen sichtbar zu machen.

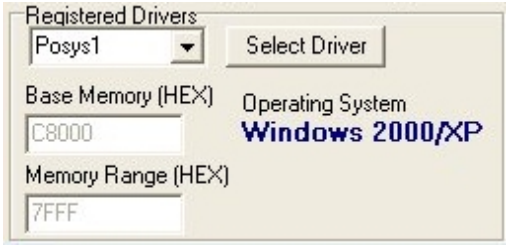
¤

Initialisierung

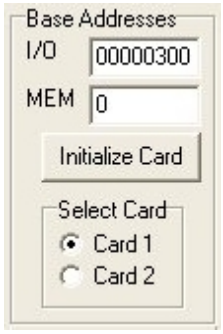
Initialisierung

Links unten im Hauptfenster ist eine Box "Registered Drivers". Je nachdem wieviele Karten (max. 16) Sie im System angemeldet haben, erscheinen hier mit einem Klick auf die Combo List Box alle angemeldeten Karten. Angenommen es

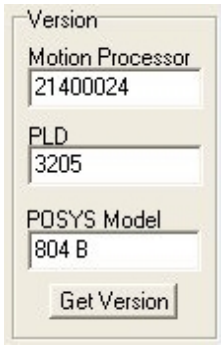
wurde nur eine Karte angemeldet, dann erscheinen 2 Möglichkeiten zur Auswahl, "None" und "Posys1". Standard ist "None", weswegen jetzt "Posys1" ausgewählt werden und im Fenster sichtbar sein sollte. Ein weiterer Klick auf den Button "Select Driver" öffnet den Treiber und reserviert den erforderlichen Adressraum für die gewählte Karte.



In der Auswahlbox "Base Addresses" wird die für die gewählte Karte reservierte Basisadresse angezeigt. In diesem Fall ist es die Adresse 0x300h.



Um nun die Kommunikation mit der Karte zu starten ist es erforderlich in der Auswahlbox "Base Addresses" den Button "Initialize Card" an zu klicken. Dies hat eine Reihe weiterer Kommandos intern zur Folge die unter anderem die Kommunikation mit der Karte eröffnen und einige Basiskommandos senden um z.B. die Kartenversion bzw. unterstützte Motorversion abzufragen. Das Ergebnis ist unter anderem in der Box "Version" zu sehen. In der ersten Reihe unter "Motion Processor" wird eine Nummer ausgegeben die den verwendeten Motion Prozessor (2140) und die Softwareversionsnummer (0024) ausgibt. Ein weiterer Klick in dieser Box auf den Button "GetVersion" liefert zudem noch die PLD Versionsnummer zurück und alles zusammen die daraus resultierende Kartenversion (POSYS Model).



Sollte bis hierher alles glatt gelaufen sein, dann wurde die Kommunikation mit der POSYS® erfolgreich aufgebaut!

¤

Beschreibung der weiteren Programmfunktionen

Beschreibung der weiteren Funktionen

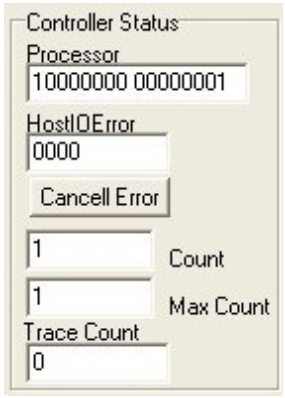
Im Folgenden werden die weiteren Fenster und Boxen und deren Bedeutung näher erläutert.

¤

Controller Status Box

Controller Status Box

Diese Box liefert Informationen bezüglich des Status des Motion Prozessors, über evtl. aufgetretene HostIO Fehlermeldungen, Counts (sollte immer Null sein, welche aber mit "1" wiedergegeben wird; ausser bei Karten mit Navigator Prozessor kleiner Versionsnummer 2.0) und die Anzahl Trace Counts bei/nach einer Datenerfassung.

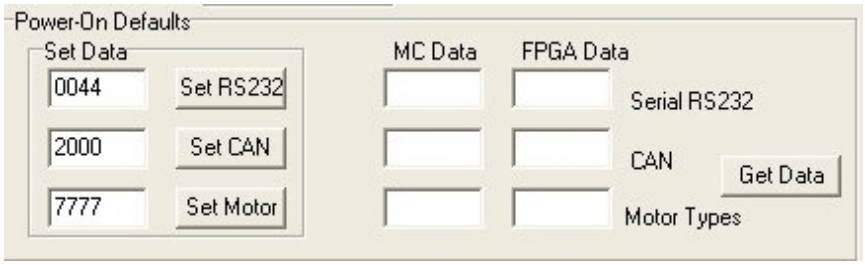


¤

Power-On Defaults Box

Power-On Defaults Box

Für die POSYS® Karten werden hier zusätzliche Informationen, die nach dem Einschalten standardmäßig geladen werden, angezeigt. Diese Einstellungen können hier überschrieben werden. Wie z.B. die Geschwindigkeit für den CAN-Übertragungsmodus, Motortype oder die Parameter für die serielle Schnittstelle. Je nachdem welche Karte Sie haben sind bestimmte Einstellungen veränderbar oder auch nicht.



¤

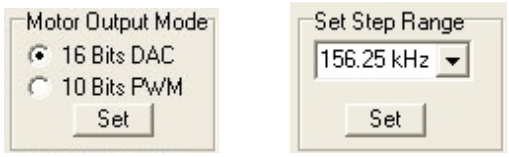
Motor Output Mode Box

Motor Output Mode Box

Je nachdem welche Art Karte installiert bzw. welche Achse aktiv ist, gibt es die Möglichkeit verschiedene Motor-relevante Einstellungen vorzunehmen. Bei Servomotoren kann gewählt werden zwischen analogem 16-Bit Ausgang oder PWM-Ausgangsmodus.

Weitere Einstellungen für 2- und 3-phasige Servomotoren sind über das Kommando Set/GetMotorType und Set/GetPWMFrequency auswählbar.

Bei Schrittmotoren kann zwischen 4 verschiedenen Step Ranges gewählt werden. Je nachdem wie hoch die maximale Ausgabefrequenz sein soll, sollte hier der entsprechend darüber liegende Wert eingestellt sein.

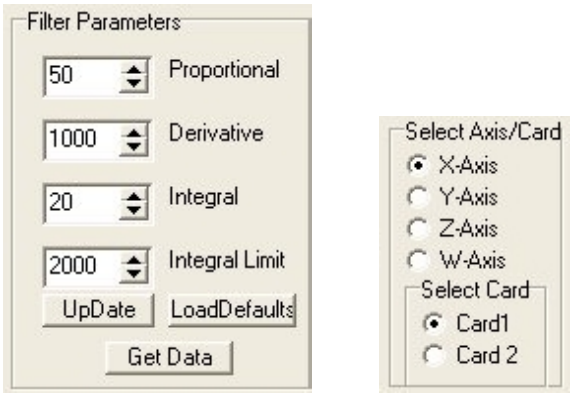


¤

Filter Parameter Box

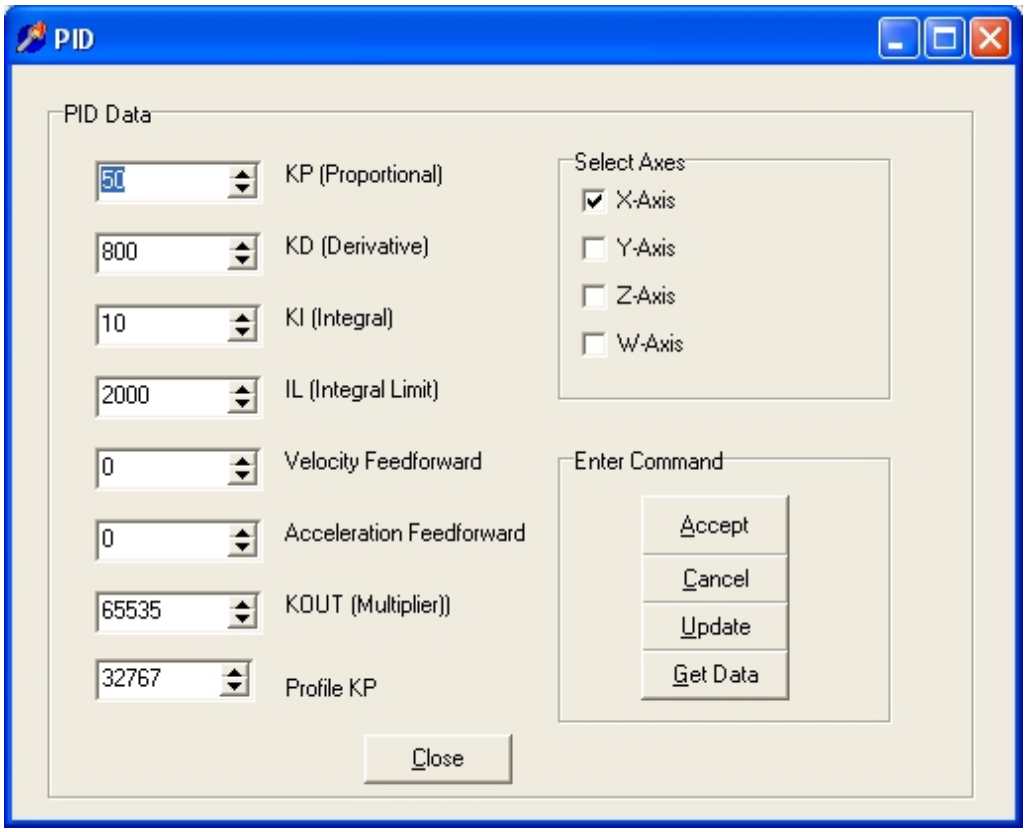
Filter Parameter Box

Je nachdem ob Servomotoren- oder Schrittmotorenkarten, oder Achsen im Servomodus oder Schrittmotormodus sich befinden ist diese Box für die Einstellung der Filterparameter zugänglich (Servo) oder nicht (Schritt). Im Servomodus werden beim Start des Programmes diese Werte standardmäßig geladen. Ein Klick auf "Load Defaults" lädt neue Werte die auch sofort für alle Achsen aktiv werden. Bei einem Klick auf den Button Update werden die im Fenster sichtbaren Werte nur für die jeweilig aktive Achse geladen. Ein Klick auf "Get Data" zeigt die Werte für die jeweilig aktive Achse an. Die Auswahl der aktiven Achse erfolgt über die Box "Select Axis/Card", wie im 2. Bild dargestellt. Weitere Filtereinstellungen und Informationen hierzu finden Sie im Programmierhandbuch zum Thema Digital Servo Filter.



Ob diese neuen Werte für Ihre Applikation/Motoren verwendbar sind, muss jeder für sich selber herausfinden. POSYS Motion Control GmbH & Co.KG kann keine Verantwortung für evtl. nicht passende Werte übernehmen, die auch zu unvorhersehbarem Verhalten der Motoren führen können. Daher wird dringend empfohlen diese neuen Werte ohne angeschlossene Motoren zuerst zu überprüfen und auf Verwendabrkeit mit Ihren Motoren zu prüfen.

Mehr Einstellmöglichkeiten bietet das über das Menü zugängliche PID-Filtereinstellmenü. Ein Klick auf PID -> PID Filter öffnet folgendes Fenster:



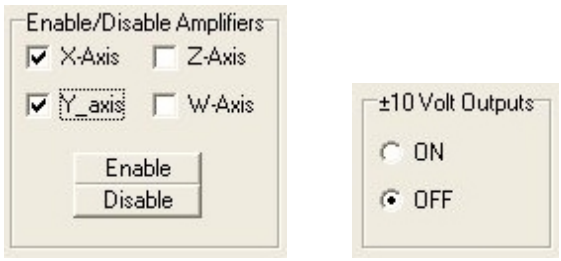
Hier können für jede Achse individuell alle relevanten Filterparameter eingestellt werden.

¤

Verstärkerfreigabe und DACOutputs ON/OFF

Verstärkerfreigabe und DACOutputs ON/OFF

Wie bei den Filterparametereinstellungen sind auch die Verstärkerfreigaben und DACOutputs ON/OFF Funktionen nur für Servokarten/Servoachsen relevant, wobei noch für die DACOutputs folgende weitergehende Einschränkung gilt, dass sie nur für Servoachsen mit analogem Motorausgangssignal von Bedeutung sind. Servoachsen im PWM-Modus sind von den DACOutputs ON/OFF Funktionen nicht betroffen.



Die Verstärkerfreigabebox erlaubt die selektive Auswahl welche der Achsen freigegeben werden sollen. Bei Achsen im PWM-Ausgabemodus ist dann sofort die Regelung aktiv. Bei Achsen in 16-Bit analogem Modus ist intern die Regelung zwar aktiv aber die Motoren werden solange nicht in der Regelung sein bis die DACOutputs auf ON geschaltet wurden. Dies stellt ein zusätzliches Sicherheitfeature dar.

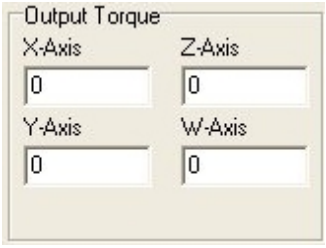
Bei Schrittmotorkarten bzw. Achsen im Schrittmotormodus können die Verstärkerfreigabesignale als zusätzliche kunden-/anwendungsspezifische Ausgänge verwendet werden. Dies ist grundsätzlich auch für Servokarten/-achsen möglich falls eine externe Schaltung für die Verstärkerfreigaben verwendet wird.

¤

Output Torque Box

Output Torque Box

Die Ausgabe bestimmter Werte in diesem Fenster sind wiederum nur für Servokarten/-achsen von Bedeutung. Für jede Achse wird individuell der momentan ausgegebene Wert für die Regelung der Servoachse(n) angezeigt. Dies geschieht sowohl im analogem Ausgabemodus als auch im PWM-Ausgabemodus.

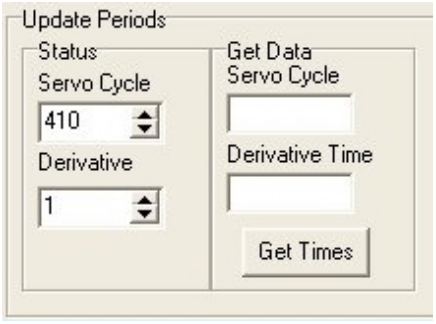


¤

Update Periods Box

Update Periods Box

Diese Box erlaubt die Abfrage der eingestellten Zykluszeit bzw. die Änderung derselbigen. Der Standard nach dem Einschalten ist immer die kürzest mögliche Zykluszeit für die jeweilige Karte und wird hier angezeigt.



Um andere Werte einzustellen kann über die entsprechenden Pfeile der Wert verändert werden, dabei werden aber nur für die jeweilig eingesetzte Karte gültige Werte angezeigt und akzeptiert.

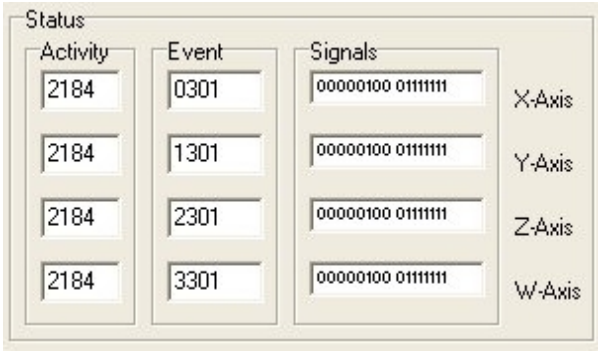
Ein Klick auf GetTime zeigt die aktuell eingestellten Werte an.

¤

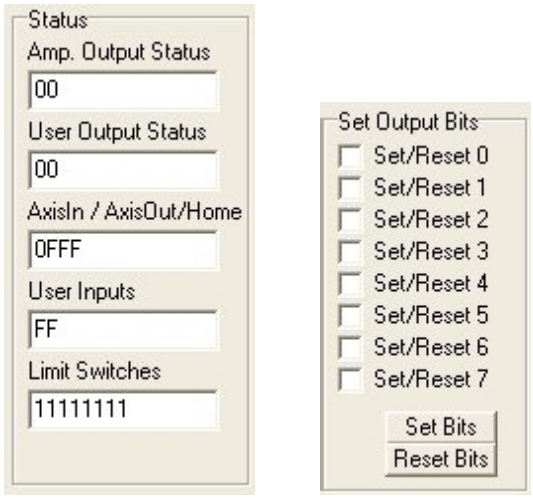
Status Activity/Event/Signal

Status Activity/Event/Signal

In dieser Box werden kontinuierlich bestimmte Aktivitäts-, Zustands- und Signalstatusmeldungen angezeigt.



z.B. kann man unter Activity sehen ob eine Achse in Bewegung ist, oder unter Signals erkennt man welchen Status die Enkoder Eingangssignale gerade einnehmen und unter Event ob eine Achse auf einem Endschalter steht oder nicht. Weitere Informationen und Anzeigen liefern hierzu auch die beiden Boxen "Status" und "Set Output Bits" (ReadIO/WriteIO), die bei Veränderung bestimmter Aktivitäten oder Zustände diese getrennt anzeigen. Aufgrund bestimmter Zustände lassen sich weitere Aktionen definieren auf die Ihre Applikation entsprechend reagieren soll. Durch die Definition von Breakpoints ist es möglich auf bestimmte Zustände und Ereignisse gezielt zu reagieren.

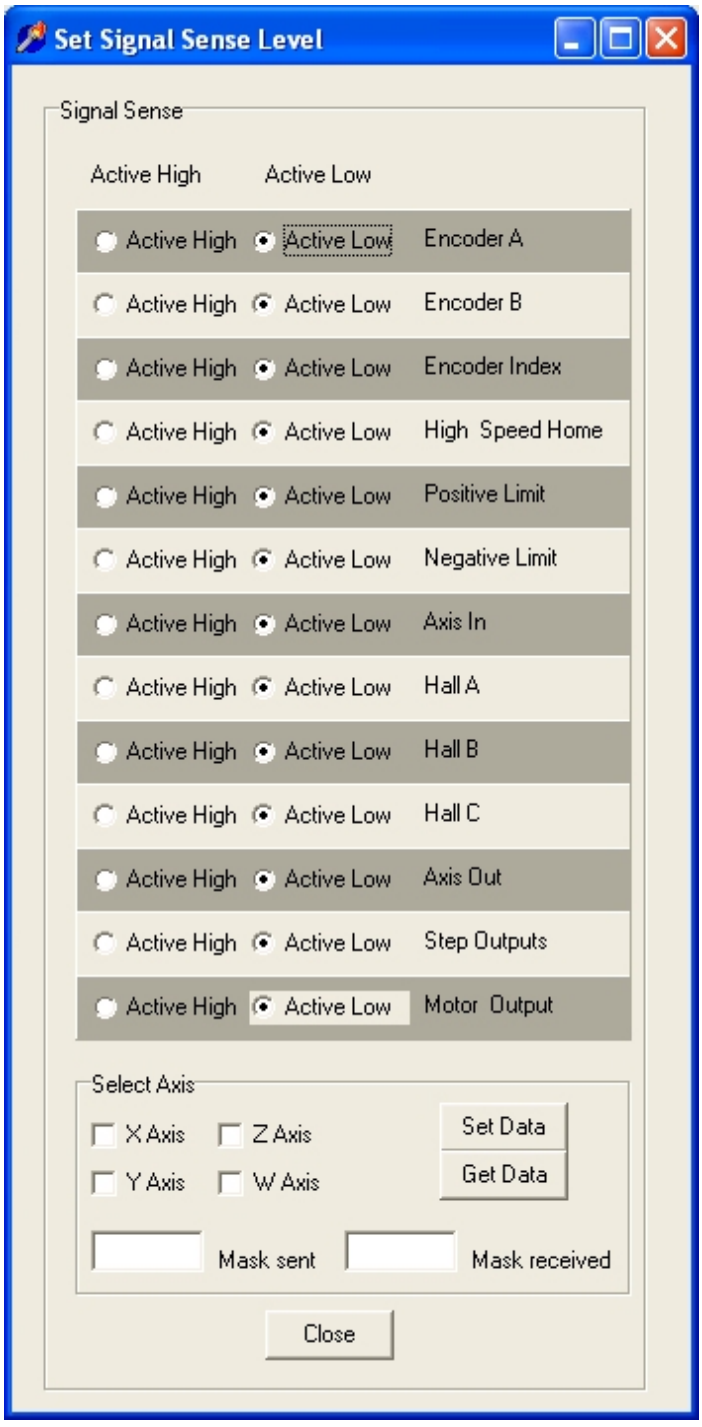


¤

Signal Sense

Signal Sense

Eine weitere Funktion der POSYS® ist die Möglichkeit zu definieren ob Signale "active low" oder "active high" interpretiert werden sollen. Dazu kann im Menü unter Aufruf von "Signal Sense -> Set Signal Sense" ein neues Fenster geöffnet werden in dem für jedes Signal definiert wird wie es von der POSYS® interpretiert werden soll.

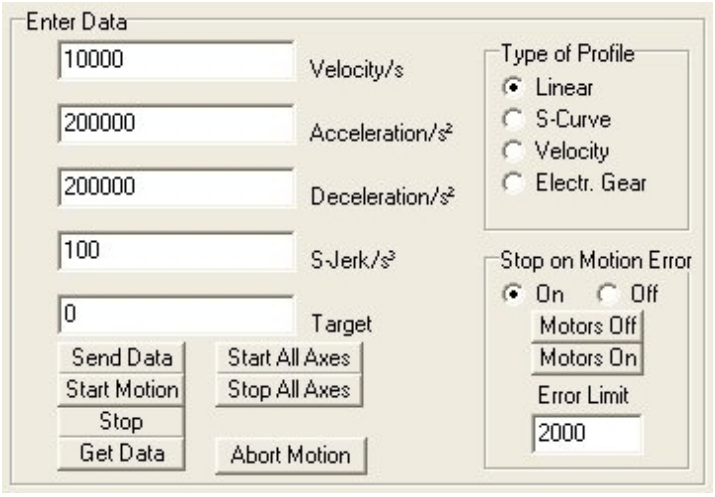

Set Signal Sense Level
Signal Sense
Active High
Active Low
Active High Active Low Encoder A
Active High Active Low Encoder B
Active High Active Low Encoder Index
Active High Active Low High Speed Home
Active High Active Low Positive Limit
Active High Active Low Negative Limit
Active High Active Low Axis In
Active High Active Low Hall A
Active High Active Low Hall B
Active High Active Low Hall C
Active High Active Low Axis Out
Active High Active Low Step Outputs
Active High Active Low Motor Output
Select Axis
X Axis
Z Axis
Y Axis
W Axis
Set Data
Get Data
Mask sent
Mask received
Close

¤

Enter Data Box

Enter Data Box

Um eine Bewegung zu programmieren sind neben den Filterparametern für Servomotoren bestimmte weitere Parameter notwendig. In dieser Box werden Geschwindigkeit, negative und positive Beschleunigung, Zielposition und optional Jerk definiert.

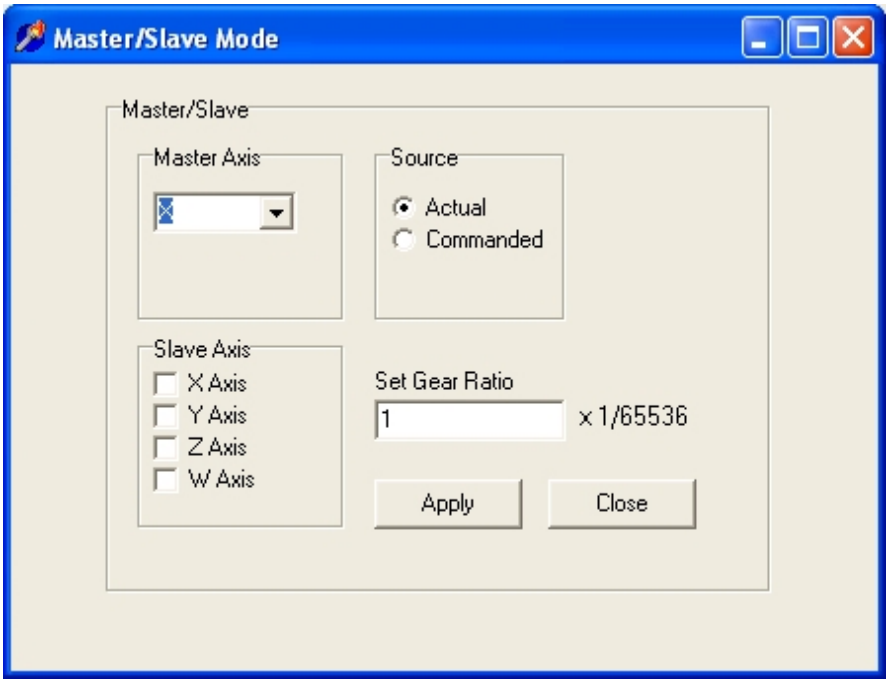


Weitere Parameter, die programmiert werden können, sind das Bahnprofil (Linear, S-Kurve, Geschwindigkeit, Electronic Gearing und (nicht angeführt) External Profile Mode) und der Schleppabstandsfehler.

Nach Eingabe der gewünschten Werte können diese für die jeweilig aktive Achse durch ein Klick auf "Send Data" übernommen werden (siehe Filter Parameter Box weiter oben). Ein Klick auf "Start Motion" ist gleichbedeutend einem Update für die jeweilig aktive Achse und würde die programmierte Bewegung starten. Sollen erst weitere Achsen mit diesen oder anderen Werten vorher programmiert werden, muss vorher die zu programmierende Achse aktiviert werden. Ein Klick auf "Start All Axes" startet alle vorprogrammierten Achsen.

Um eine Achse zu stoppen, muss sie als aktiv markiert sein und ein Klick auf "Stop" würde augenblicklich die Achse mit Rampe stoppen. Um alle Achsen anzuhalten reicht ein Klick auf "Stop All Axes". Ein Klick auf "Abort Motion" würde alle Achsen augenblicklich und ohne Rampe stoppen.

In der rechten Box "Type of Profile" kann ausgewählt werden welche Art von Bewegungsprofil aktiv sein soll. Bei der Auswahl von S-Kurve muss auch der Wert für "S-Jerk/s³" angepasst werden. Bei der Auswahl von "Electronic Gearing" können zusätzlich noch über das Hauptmenü "Options -> Master/Slave Mode" weitere Einstellungen vorgenommen werden.



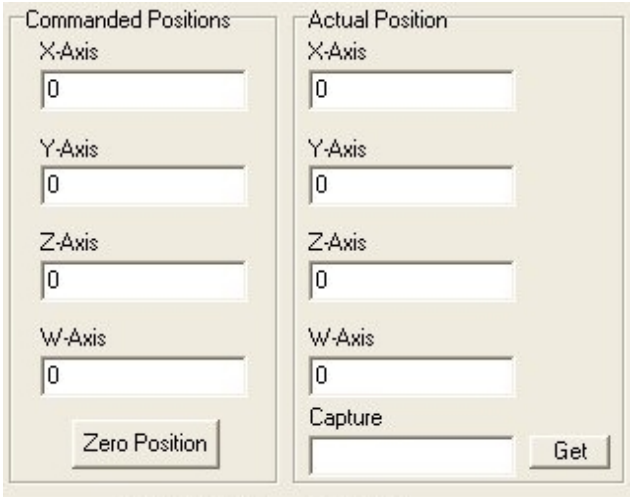
Dabei stehen zur Auswahl welche Achse(n) als Master und welche Achse(n) als Slave fungieren, ob nach der Sollposition oder Istposition geregelt werden soll und in welchem Verhältnis jede Slave-Achse der Master-Achse folgen (positiv/negativ, etc.) soll.

¤

Commanded und Actual Position Anzeigebox

Commanded und Actual Position Anzeigebox

Natürlich ist auch die Ausgabe der Soll- und Istposition zu einem gegebenen Zeitpunkt notwendig. Für diesen Zweck sind die beiden Boxen "Commanded Position" und Actual Position" gedacht.



Nach Eingabe eines Bewegungsprofils und Update der Parameter würde eine Bewegung initiiert werden. Anhand der eingegebenen Parameter errechnet der Controller für jede Updaterate eine Sollposition und versucht durch Ausgabe eines entsprechenden Drehmomentsollwertes für Servomotoren oder Takt- & Richtunggsignales für Schrittmotoren möglichst den Istwert mit dem Sollwert gleich zu setzen. Einschränkend sei hierzu gesagt, dass eine Regelung für Schrittmotoren nicht stattfindet. Falls Schrittmotoren mit Enkoder eingesetzt werden, ist aber der Controller in der Lage über die Funktion Stall Detection einen möglichen Motorstillstand bzw. Verlust an Takten zu erkennen. Es liegt im Verantwortungsbereich des Anwenders dann entsprechende Maßnahmen zu ergreifen, Taktverluste auszugleichen. Ob diese Position auf der Istseite gleich oder unterschiedlich der Sollposition ist, hängt maßgeblich von den eingegebenen Parametern ab. Generell kann angemerkt werden, dass je steifer die Einstellung ist desto identischer werden Soll- und Istwert sein.

Der Button "Zero Position" erlaubt für die aktivierte Achse die Position auf Null zu stellen.

Im Ausgabefenster "Capture" und durch ein Klick auf den Button "Get" kann die Indexposition angezeigt werden, die immer einmal pro Umdrehung erfasst werden kann. Auch dies funktioniert immer nur für die jeweilig aktivierte Achse und nur wenn der Enkoder über einen Index verfügt.

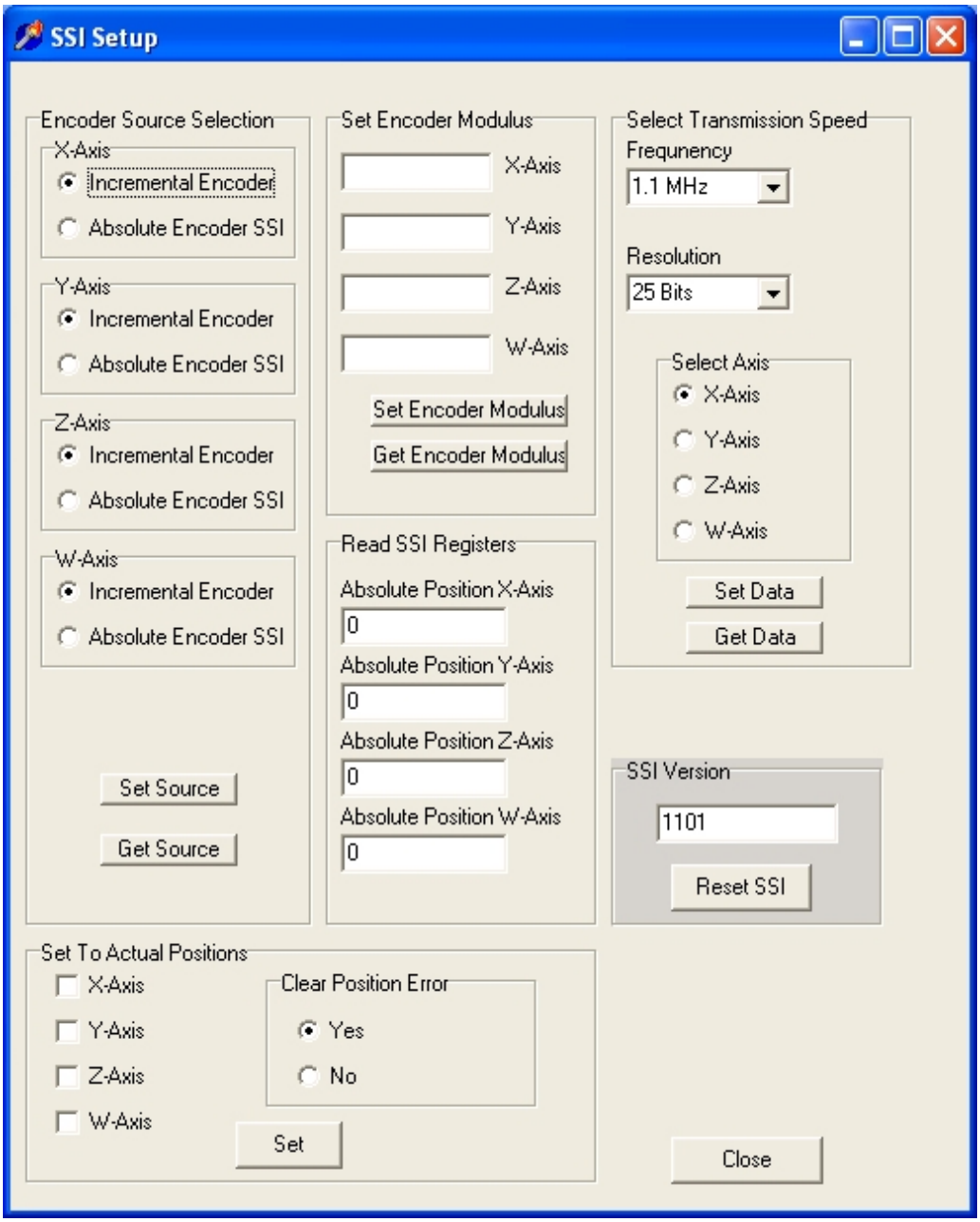
¤

SSI Rückmeldung

SSI Rückmeldung

Standardmäßig erlauben die POSYS® Karten die Verwendung von inkrementalen Rückmeldeeinheiten mit und ohne invertierten Signalen. Als Zusatz können auch Absolutwertgeber mit SSI-Schnittstelle verwendet werden. Dabei ist es möglich Absolutwertgeber mit Gray Code oder Binär Code einzusetzen.

Falls die SSI-Option verwendet wird, erlaubt die POSYS® GUI Software Einstellungen vorzunehmen, die auch die Jumpereinstellungen auf dem Optionenboard überschreiben. Mehr hierzu finden Sie im Kapitel über die SSI800/900 Optionenkarte.

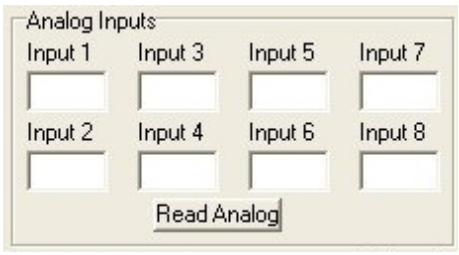


Jede Achse kann individuell auf die eine oder andere Rückmeldeeinheit eingestellt werden. Während des Betriebes, dabei sollten die Motoren still stehen oder sogar ausgeschaltet sein, kann auch zwischen inkrementalen und Absolutwertgebern hin- und hergeschaltet werden, sofern beide Enkodertypen eingesetzt werden. Es ist aber unbedingt darauf zu achten, dass mögliche Unterschiede in den Soll-/Istwertangaben vorher mit einem ClearPositionError abgeglichen werden. Weitere Informationen hierzu finden Sie im Kapitel Motoranbindung und im Programmierhandbuch zu den Kommandos Enkoder.

¤

Analog Inputs

Die POSYS® verfügt über 8x 10-Bit analoge Eingänge. Die Erfassung und der Wert der dort gelesen wird, kann mit dem Kommando ReadAnalog erfolgen. Je nachdem welche POSYS® eingesetzt wird, sind die möglichen Eingangswerte 0-2.048VDC, 0-2.5VDC oder 0-4.096VDC. Für weitere Informationen bzgl. der erlaubten Spannungswerte lesen Sie bitte nach im Benutzerhandbuch in der Sektion POSYS® Versionsspezifikationen.


Analog Inputs
Input 1
Input 3
Input 5
Input 7
Input 2
Input 4
Input 6
Input 8
Read Analog

¤

Single Port RAM / Dual Port RAM

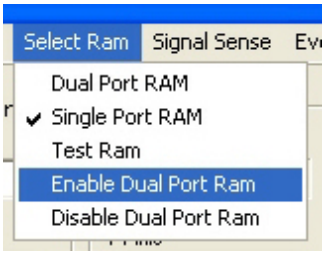
Single Port RAM / Dual Port RAM

Die POSYS® Positioniersteuerkarten verfügen je nach Modell über Single Port RAM (256k POSYS® 700/800, 512k POSYS 800-B/900/1800/1900) oder optional 1024k). Die Modelle POSYS® 1800/1900 sind je nach Modell optional oder standardmäßig mit 16k x 16 oder 32k x 16 Dual Port RAM bestückt.

Die folgende Tabelle zeigt auf welche Karten mit welchem RAM-Typ standardmäßig bestückt sind und welche Alternativen dazu bestehen.

POSYS® model	256 KB SPRAM	512 KB SPRAM	16 KB x 16 DPRAM	32 KB x 16 DPRAM	64 KB x 16 DPRAM
700/800	standard	n/a	n/a	n/a	n/a
800-B/900	option	standard	n/a	n/a	n/a
1800	option	option	standard	option	option
1900/1950	option	option	option	standard	option
1920	option	option	option	standard	option

Die POSYS® GUI erlaubt entsprechend der Ausstattung den Zugriff entweder nur auf Single Port RAM oder bei Modellen mit Dual Port RAM den Zugriff über Single Port RAM Mode oder Dual Port RAM Mode auf den On-Board Speicher.



Anmerkung: POSYS® 1800

Für die POSYS® 1800 muss für den Dual Port RAM Mode der Speicher zuerst freigegeben werden. Dies hat nichts mit der Adressdefinition des Speicherbereiches zu tun, sondern betrifft den Modus wie auf den Speicher zugriffen wird. Im Single Port RAM Mode kann man auf Dual Port RAM über den Motion Control Prozessor zugreifen als wäre es nur Single Port RAM. Dies kann evtl. hilfreich sein, falls ein PC-System den zugewiesenen Speicherbereich für die POSYS® nicht reservieren kann oder der zu reservierende Speicherbereich erst noch ermittelt werden und nicht gleich ein Reboot erfolgen soll. Im Dual Port RAM Mode wird direkt von Bus-Seite auf den Speicher zugegriffen.

Desweiteren ist der Dual Port RAM nach dem Einschalten nicht aktiviert. Im Menü muss bevor jeglicher Zugriff auf Dual Port RAM im Dual Port RAM Mode erfolgen kann, dieser über den Menüpunkt "Select Ram -> Enable Dual Port Ram" aktiviert und anschliessend der Modus "Dual Port Ram" ausgewählt werden. Um den Dual Port RAM Mode wieder zu deaktivieren, genügt es den Menüpunkt "Disable Dual Port Ram" auszuwählen.

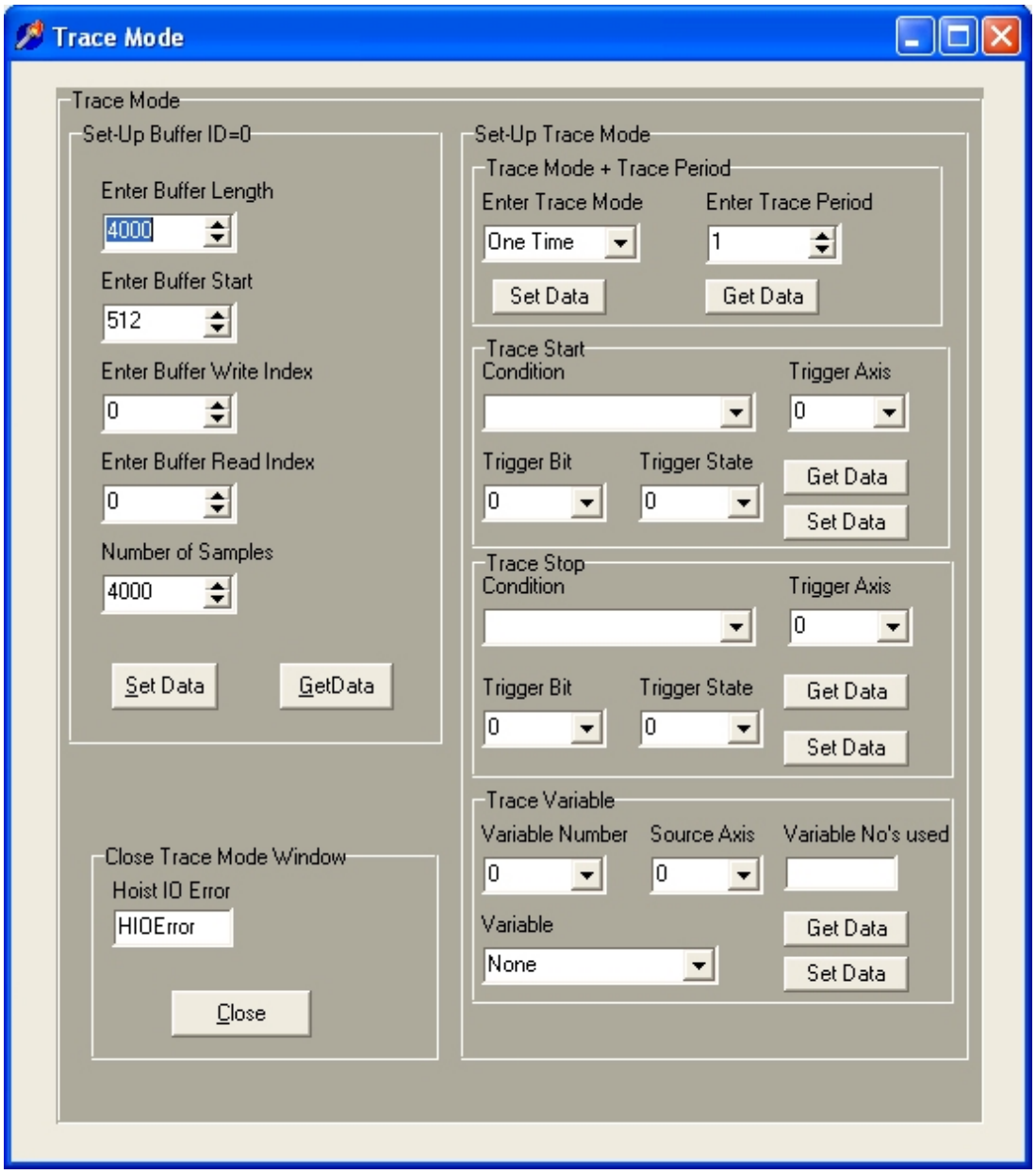
Diese Punkte müssen bei der Verwendung einer eigenen Softwareapplikation mitbedacht und entsprechend intergriert werden. Mehr zu diesem Thema finden Sie im Kapitel Verwendung des On-Board Dual Port RAMs mit der POSYS® 1800.

¤

Trace Mode

Trace Mode

Der Trace Modus erlaubt die Verwendung des On-Board Speichers zur Zwischenspeicherung bestimmter erfasster Daten bzw. kann er verwendet werden um vorkalkulierte Bahndaten abzuspeichern, die der Controller nach und nach abarbeitet ohne weitere Hostintervention. Zu erreichen ist diese Funktion über den Menüpunkt "Options -> Trace Mode".



Das obige Fenster zeigt eine Anordnung die hauptsächlich für Diagnostikzwecke ausgelegt ist. Um den On-board Speicher zu verwenden muss er zuerst initialisiert werden. Dazu müssen im linken oberen Fenster die Bufferlänge, der Bufferstartpunkt und die Anzahl zu erfassende Samples angegeben sein. Bei POSYS® Karten mit Navigator Chipsatz (POSYS® 700/800/800-B/900) ist der kleinste Bufferstartpunkt 512. Bei den Magellan-basierten POSYS® 1800/1900 kann der Startpunkt auch mit Null angegeben werden. Die Bufferlänge kann und darf nicht den maximal zur Verfügung stehenden physikalisch vorhanden Speicher überschreiten.

Im Fenster "Set-Up Trace Mode" werden weitere Einstellungen zur Erfassung vorgenommen. Angegeben werden müssen noch Trace Mode & Trace Period. Der Trace Mode legt fest ob eine kontinuierliche oder Einzelerfassung stattfinden soll. Bei einer kontinuierlichen Erfassung würde der Controller beim Erreichen des Endes des definierten Buffers wieder von vorne anfangen Daten zu hinterlegen. Dabei werden vorher erfasste Daten wieder überschrieben. Falls diese Daten später ausgewertet werden sollen, müssen Sie vorher ausgelesen und abgespeichert werden. Bei einer Einzelerfassung wird nur einmal der definierte Speicherbereich mit erfassten Daten beschrieben und die Erfassung endet.

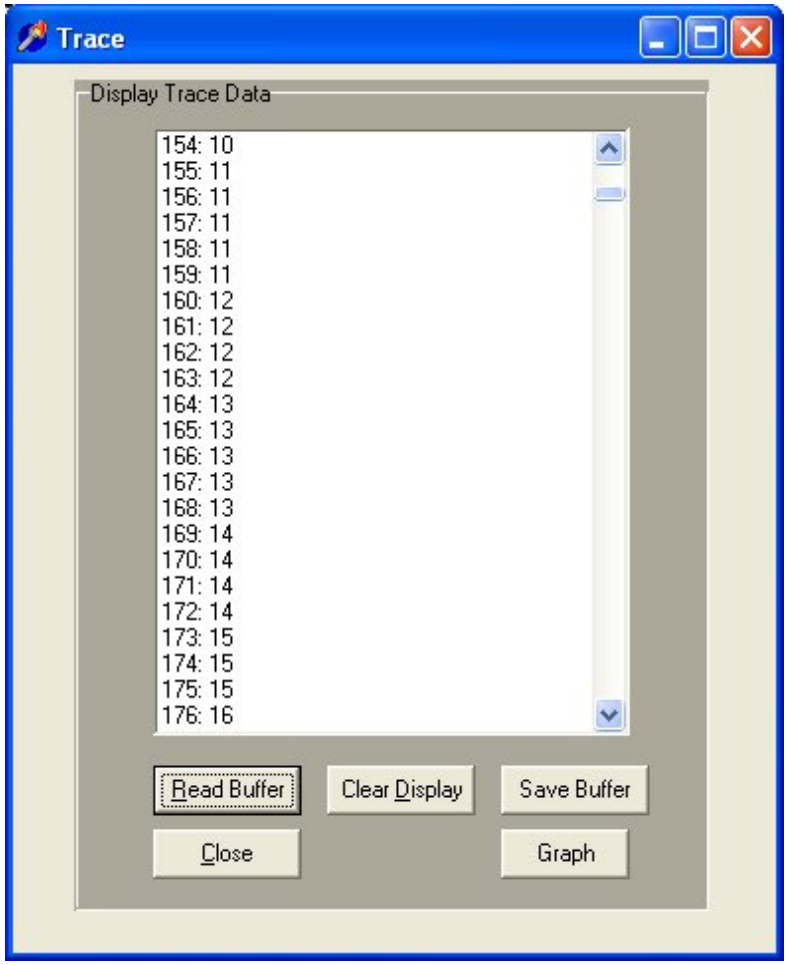
Im Fenster "Trace Start" kann festgelegt werden unter welchen Bedingungen eine Erfassung beginnen soll.

Im Fenster "Trace Stop" kann optional festgelegt werden unter welchen Umständen eine Erfassung beendet werden soll.

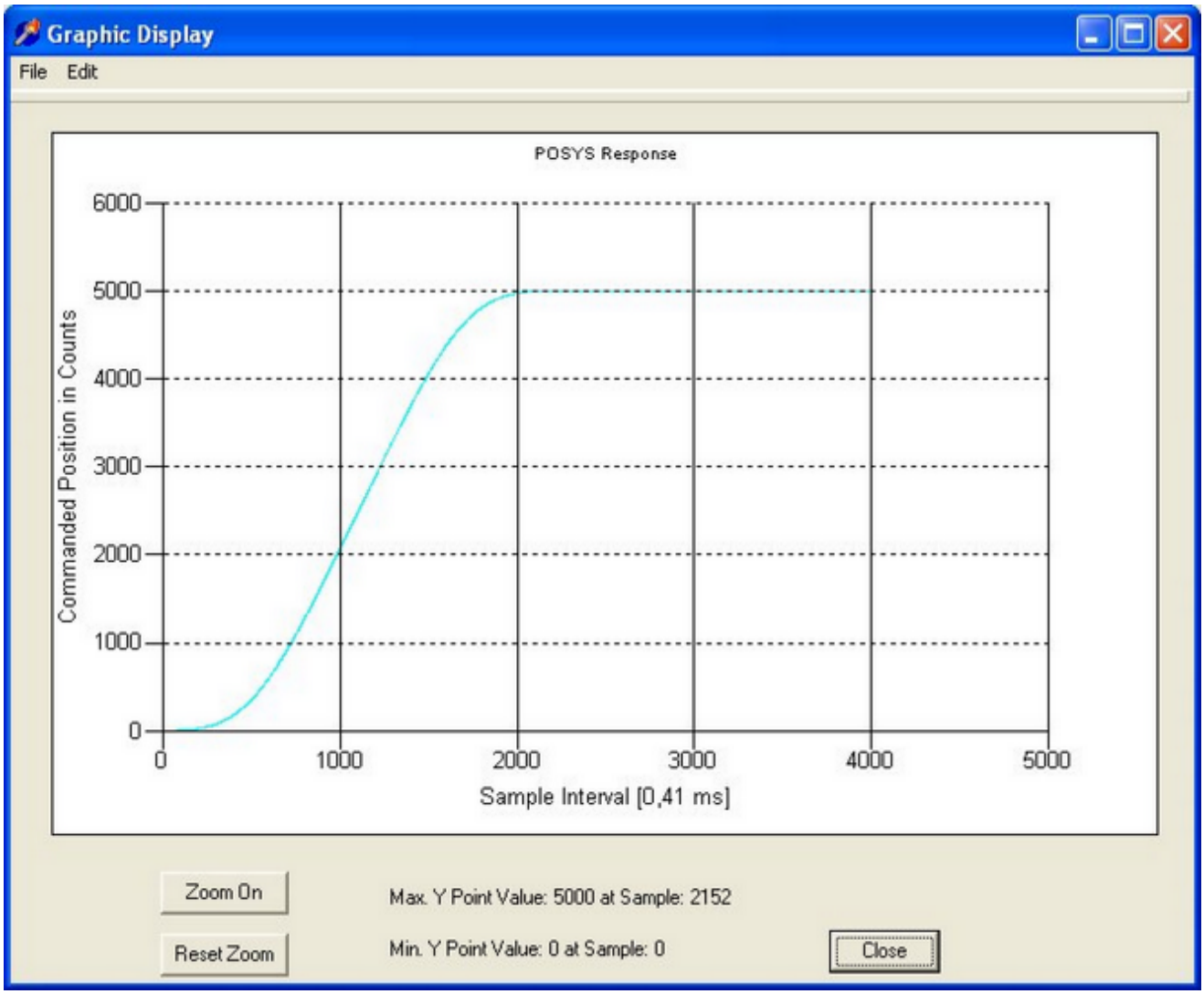
Im Fenster "Trace Variable" wird definiert welche Informationen gesammelt werden sollen. Dabei muss als letzte Varable immer "None" mit ausgewählt werden, sofern nicht insgesamt 4 Variablen zur Erfassung ausgewählt wurden, auch wenn nur eine Variable erfasst werden soll muss "None" als 2. Variable definiert sein.

Ein klick auf "Close" schliesst dieses Fenster wieder.

Über das Menü "Options -> Display Trace Data" öffnet sich folgendes Fenster. In diesem Fenster lassen sich die erfassten Werte aus dem Speicher einlesen (Button "Read Buffer"), mit "Clear Display" werden die Werte aus dem Fenster gelöscht (nicht aus dem On-Board Speicher, dort sind sie weiterhin vorhanden), mit "Save Buffer" können die erfassten Daten in einer Datei gespeichert werden.



Nach einer Erfassung können die gesammelten Werte im Fenster "Graphic Display" visualisiert werden. Dazu auf den Button "Graph" klicken und folgendes Fenster wird geöffnet:



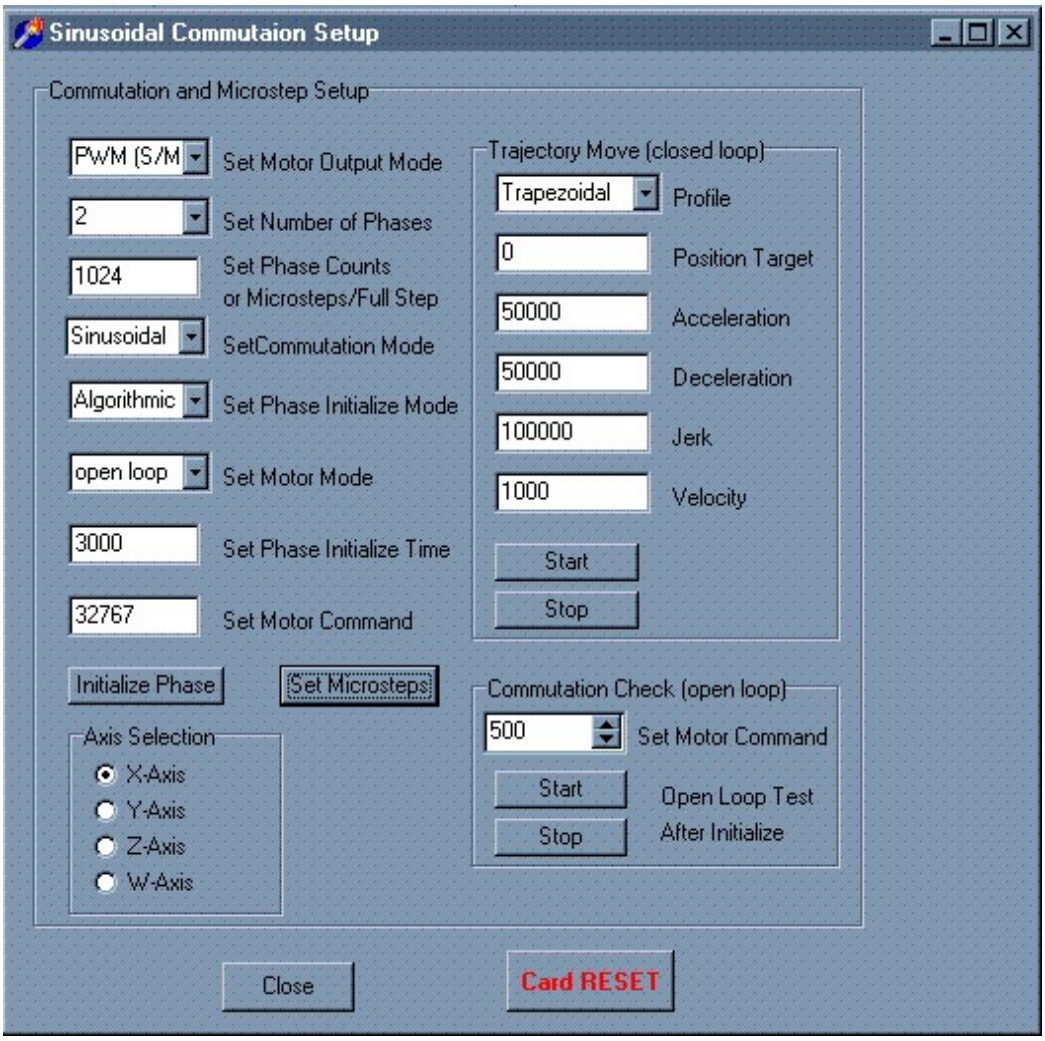
Mit einem Klick auf "Zoom On" kann die Zoom-Funktion eingeschaltet werden. Dazu anschliessend auf der Fläche mit der rechten Maustaste klicken und die auf zu ziehende Fläche markieren. Nach dem Loslassen wird der markierte Bereich im Fenster angezeigt und evtl. vorher nicht sichtbare Details sind jetzt sichtbar. Ein Klick auf den Button "Reset Zoom" setzt die Bildschirmausgabe wieder zurück. Ein Klick auf "Close" schliesst das Fenster wieder.

Im Menü (File - Edit) sind noch Funktionen aufrufbar um die Grafik auf einem Drucker auszugeben oder in einer Datei abzuspeichern.

¤

Setupmenü für Sinuskommutierung und Mikroschrittmotormodus

Über das Menü "Commutation/Microstep" wird folgendes Fenster geöffnet. Diese Fenster ist nur dann zugänglich wenn eine entsprechende Karte verwendet oder eine Achse im entsprechenden Modus eingesetzt wird. Dies sind die Modi für bürstenlose Servomotoren mit interne Kommutierung oder Mikroschrittmodus.



Auf der linken Seite werden die Signalausgabe, Anzahl Phasen etc. eingestellt. Links unten wird definiert für welche Achse die Werte gelten sollen. Für weitergehende Informationen verweisen wir auf die Technical Note (FAQ) Sinus Kommutierung und MikroschrittMotorModus Setup.

¤

Index

- 1 -

16 Bit DAC Signalausgabe 170

- 5 -

50/50 Impulsbreiten Modulierte Signalausgabe 170

- A -

AC Induction Motor Control 201
Activity Status Register 139
AdjustActualPosition 276
Aktivieren und Deaktivieren von Control Modules 100
Aktuelle Positionsregister 159
Analog Eingang 157
Analog Eingang Pinout Beschreibung 65, 77, 54, , , 39
Analog Eingang Pinout Beschreibung 65, 77, 54, , , 39
Analog Eingang Pinout Beschreibung 65, 77, 54, , , 39
Analog Inputs 477
AtRest Indikator 197
Auswahl vom 1-Phasen Signalausgabemodus mit der POSYS® 183
AxisIn Input 157
AxisOut Output 156

- B -

Bahnen, Profile und Parameter 103
Beschreibung der weiteren Programmfunktionen 468
BiQuad Ausgangsfilter 129
BitReset 388
BitSet 389
BitTest 390
Breakpoint Aktionen 123
Breakpoint Beispiele 124
Breakpoint Trigger 122
Breakpoints 122, 406
Bürstenbehaftete Servomotor Pinout Beschreibung 71, 47, 62, 36
Bürstenbehaftete Servomotor Pinout Beschreibung 71, 47, 62, 36
Bürstenlose Servo- und Mikroschrittmotor Pinout Beschreibung 49
Bürstenlose Servo- und Mikroschrittmotor Pinout Beschreibung 49
Bürstenlose Servo- und Mikroschrittmotoren Pinout Beschreibung , , 72
Bürstenlose Servo- und Mikroschrittmotoren Pinout Beschreibung , , 72

- C -

CAN 2.0B 45, 69
CAN Ereignis Benachrichtigung 177
ClearInterrupt 216
ClearPositionError 235
Closed Loop and Open Loop Control Modes 133
Command Summary 201
Commanded und Actual Position Anzeigebox 476
CompareActivity 391
CompareEvent 392
CompareSignal 393
Control Flow Überblick 99
Controller Area Network CAN2.0B 176

Controller Status Box 469
Coordinated Motion with POSYS® Contouring Library 398
Coordinated Moves using POSYS® 900 Motion Controller 402

- D -

DAC und PWM Motorausgabe (POSYS® Mikroschrittmodus) 198
DACOutputs 236
Das POSYS® GUI Hauptfenster 466
Das SetStop Kommando 116
Deaktivieren und Aktivieren des Position Loop Modules 101
Definition der Parameter 103
Der SYNC IO - Multiple Controller Synchronization 157
DetectPosysDevices 214
Digital Inputs & Outputs 404
Digital IO Specification POSYS® 7xx/8xx/8xx-B/9xx 403
Digitale Filterung 160
DisableDPRam 293
Dual Encoder Support 128

- E -

Einen Breakpoint definieren; Übersicht 122
Einleitung 465
Einleitung POSYS® 18xx/19xx 13
Einleitung POSYS® 7xx/8xx-B/9xx 14
Einstellbare Komponenten , , 33, 59, 67, 41
Electronic Gear Profile 113
Elektrische Spezifikationen 95
Elektrisches Diagramm für CAN Schnittstelle 178
EnableDPRam 294
Encoder Pre-Scalar 191
Endschalter 155
Enkoderrückmeldung 194
Enter Data Box 474
Error Code 410
Event Status Register 138
Examples 450
Extension Port Connector J6 (POSYS® 700, ISA-Bus) 396
External Profile Mode 114

- F -

Filter Parameter Box 470

- G -

Geschwindigkeitsmodus 112
GetAbsoluteSSIPosition 277
GetActiveMotorCommand 305
GetActiveOperatingMode 316
GetActivityStatus 347
GetActivityStatus, Set/GetMotorCommand 321
GetActualVelocity 278
GetAmpEnable 237
GetAxis_In_Out_Home 349
GetCaptureValue 279
GetChecksum 369
GetCommandedAcceleration 325
GetCommandedPosition 326
GetCommandedVelocity 327
GetCurrentMotorCommand 305

GetDerivative 238
GetDriveStatus 350
GetEventStatus 351
GetHostIOError / GetInstructionError 370
GetIntegral 239
GetInterruptAxis 217
GetPhaseCommand 224
GetPositionError 240
GetPositionLoopValue 241
GetProcessorStatus 371
GetSignalStatus 352
GetSSIVersion 280
GetTime 338
GetTraceCount 359
GetTraceStatus 360
GetVersion 372
GP CON Steckverbinder 78

- H -

HardReset/MCReset 373
Hardware & Treiber Installation: Rapid Driver 30
Hardware & Treiber Installation: WinDriver 27
Hardware Signale; Überblick 155
High Speed Position Capture 160
Host Interrupts 142

- I -

Impressum 11
In-Motion Indikator 143
Indextakt bestimmen 185
Initialisierung 467
InitializePhase 225
InitializePOSYS 211
Inkrementaler Enkodereingang 159
Instruktionsfehler 139
IO700/800-1 79
IO700/800-2 82

- J -

Justierung der Phasenwinkel 190

- K -

Kommutierungsfehlererkennung 186
Kommutierungsparameter 185
Konfiguration der CAN2.0B Schnittstelle 177
Konfiguration der Motorsignalausgabe 192

- L -

Limit Switches 415
Liste der verfügbaren Kommandos 206
Loop Instructions 456

- M -

Memory Page Pointer 180
Mikroschritt Sinus/Cosinus Motorausgangssignale 199
Mikroschrittausgangssignale 200
Mixed Motor Mode 420

Modelle 16
Motion Complete Indikator 144
MotionScript® Utility Functions 454
Motor Bias 133
Motor Mode 117
Motor Mode Konfiguration; POSYS® 18xx 46
Motor Output Mode Box 469
Motor Treiber Konfigurationen 201
Motorausgabesignale 168
Motorsignalausgabe Interpretation 198
Motorsignalausgangskontrolle 200
Multi-Turn Systeme 161
Multiple Cards 421
MultiUpdate 317

- N -

NoOperation 376

- O -

On-Board Memory Buffer 181
On-Board Memory Read 180
On-Board Memory Write 180
On-Board RAM 412
On-Board Speicher Signal Dekodierung 180
On-Board Speicherkommandos 181
OpenDevice/CloseDevice 212
OPTON CON 86
Other MotionScript® Functions 455
Output Limit 132
Output Torque Box 472
Overview; MotionScript® 446

- P -

Parallel Device I/O 163
Parallel-word Device Interfacing 162
Paralleler Kommunikationsport 172
Parallelwort Positionsrückmeldung 160
Parameter Buffering 120
PCI ScanForPOSYS 423
Peripheral Device Read 165
Peripheral Device Write 167
Phaseninitializierung 186
PID Regelalgorithmus 126
Piezo (Keramik) Motorfunktionen 135
PMW Dekodierung 198
PortInOutW 377
Position Loop Werte lesen 102
POSYSDriverXSetup 447
POSYS® 7xx 89
POSYS® 8xx-B/18xx 90
POSYS® 9xx/19xx 91
POSYS® Functions and Procedures 458
Power-On Defaults Box 469
Program Use 449
Programmer´s Info 428
Programmierung der Phaseninitializierung 189
Pulse Generation Control 197

- R -

Read/WriteSSIFrequency 281
Read/WriteSSIRegister 282
Read/WriteSSIResolution 283
ReadAnalog 378
ReadBuffer 294
ReadDPRamLong 296
ReadIO 379
ReadIO / WriteIO 394
ReadPLD 380
RegisteredPosysDevices 215
Reserved Words 453
ResetEventStatus 353
ResetSSI 284
Resistor Networks for TTL or inverted Encoder Signals 437
RestoreOperatingMode 318

- S -

S-Kurven Punkt-zu-Punkt Profil 110
Schleppabstandsfehler 143
Schrittmotor Pinout Beschreibung , , 38, 63, 75, 52
Serial Interface 438
Serielle Schnittstelle 35, 61, 69, 45
Serieller Kommunikationsport 172
Set/GetAcceleration 328
Set/GetActualPosition 286
Set/GetActualPositionUnits 287
Set/GetAutoStopMode 244
Set/GetAuxiliaryEncoderSource 245
Set/GetAxisMode 339
Set/GetAxisOutMask 354
Set/GetAxisOutSource 356
Set/GetBiQuadCoefficient 246
Set/GetBreakpoint 218
Set/GetBreakpointUpdateMask 220
Set/GetBreakpointValue 221
Set/GetBufferFunction 297
Set/GetBufferLength 298
Set/GetBufferReadIndex 299
Set/GetBufferStart 300
Set/GetBufferWriteIndex 301
Set/GetCANMode 381
Set/GetCaptureSource 288
Set/GetCommutationMode 226
Set/GetDeceleration 329
Set/GetDerivativeTime 248
Set/GetDiagnosticPortMode 382
Set/GetEncoderModulus 289
Set/GetEncoderSource 290
Set/GetEncoderToStepRatio 291
Set/GetEventAction 319
Set/GetGearMaster 330
Set/GetGearRatio 331
Set/GetHoldingCurrent 307
Set/GetIntegrationLimit 249
Set/GetInterruptMask 223
Set/GetJerk 332
Set/GetKaff 250
Set/GetKd 251

Set/GetKi 252
Set/GetKout 253
Set/GetKp 254
Set/GetKvff 255
Set/GetLimitSwitchMode 340
Set/GetMotionCompleteMode 341
Set/GetMotorBias 308
Set/GetMotorCommand 309
Set/GetMotorLimit 256
Set/GetMotorMode 310
Set/GetMotorType 311
Set/GetNumberPhases 227
Set/GetOperatingMode 321
Set/GetOutputMode 312
Set/GetPhaseAngle 228
Set/GetPhaseCorrectionMode 229
Set/GetPhaseCounts 230
Set/GetPhaseInitializeMode 231
Set/GetPhaseInitializeTime 232
Set/GetPhaseOffset 233
Set/GetPhasePrescale 234
Set/GetPosition 333
Set/GetPositionErrorLimit 257
Set/GetPositionLoop 258
Set/GetProfileKp 261
Set/GetProfileMode 334
Set/GetPWMFrequency 313
Set/GetpzIntLim 263
Set/GetpzIntLimStl 264
Set/GetpzKaff 265
Set/GetpzKd 266
Set/GetpzKdPeriod 267
Set/GetpzKi 268
Set/GetpzKp 269
Set/GetpzKvff 270
Set/GetpzKvff2 271
Set/GetpzOffsetNeg 272
Set/GetpzOffsetPos 272
Set/GetpzWindow 274
Set/GetpzWindow2 275
Set/GetSampleTime 342
Set/GetSerialPortMode 383
Set/GetSettleTime 344
Set/GetSettleWindow 345
Set/GetSignalSense 358
Set/GetStartVelocity 335
Set/GetStepRange 314
Set/GetStopMode 336
Set/GetSynchronizationMode 385
Set/GetTraceMode 361
Set/GetTracePeriod 362
Set/GetTraceStart 363
Set/GetTraceStop 365
Set/GetTraceVariable 367
Set/GetTrackingWindow 346
Set/GetUpdateMask 323
Set/GetVelocity 337
SetActualToAbsolutePosition 285
SetAmpDisable 242
SetAmpEnable 243

SetEventAction Prozess 153
SetMotionCompleteMode 458
Settled Indikator 145
Setupmenü für Sinuskommutierung und Mikroschrittmotormodus 482
SetWatchDog 386
Sicherheits- & Gewährleistungshinweis 12
Sign/Magnitude Impulsbreiten Modulierte Signalausgabe 169
Signal Sense 473
Signal Sense Mask 141
Signal Status Register 140
Signalformen für die Kommutierung 184
Single Port RAM / Dual Port RAM 479
Sinusoidal Commutation and Microstep Mode Setup 418
Smoothing Factor 106
Speicherkonfiguration 179
SSI Rückmeldung 476
Stall Detection 195, 397
Status Activity/Event/Signal 472
Status Read Operation 440
SYNC IO , , 68, 45
Synchronization of Multple Cards (POSYS® 8xx-B/9xx w/ Sync-Feature and 18xx/19xx) 442
Systemüberblick 15

- T -

Takt & Richtungssignalgenerierung 196
Trace Capture 146
Trace Mode 479
Tracking Window 152
Trajectory Control Units 194
Trapezfömiger Punkt-zu-Punkt Profil 104

- U -

Update 324
Update Periods Box 472

- V -

Variables 452
Velocity, Acceleration and Jerk Calculation 444
Versionen 18xx 41
Versionen 19xx 67
Versionen 9xx 59
Verstärkerfreigabe und DACOutputs ON/OFF 471
Verwendung des On-Board Dual Port RAMs mit der POSYS® 1800 55

- W -

Wie dieses Handbuch verwendet wird 204
WriteBuffer 302
WriteDPRAMLong 303
WriteIO 387
Writing MotionScript® Functions 457

- Z -

Zykluszeit 117

- Ä -

Änderungen zu vorhergehenden Handbüchern 21

- Ü -

Überblick 126, 138, 159, 172, 183, 194, 89

© POSYS Motion Control GmbH & Co.KG, 2013
www.halbeck.com